MASTER EN SCIENCES INFORMATIQUES

Protocoles interactifs ("zero-knowledge") d'identification
Application à la carte à microprocesseur

Staelens, Yvan

Award date:
1988

Awarding institution:
Universite de Namur

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX - NAMUR

INSTITUT D'INFORMATIQUE

PROTOCOLES INTERACTIFS ("ZERO-KNOWLEDGE") D'IDENTIFICATION.

APPLICATION A LA CARTE A MICROPROCESSEUR.

Mémoire présenté pour l'obtention du grade de Licencié et Maître en Informatique

par

Yvan STAELENS

PROMOTEUR :

Mr J. RAMAEKERS.

Année Académique 1987 - 1988

Facultés Universitaires Notre-Dame de la Paix
Institut d'Informatique

rue de Bruxelles, 61, B-5000 NAMUR
Tél: 081-22.90.61 Télex: 59222 facnam-b Téléfax: 081-23.03.91

PROTOCOLES INTERACTIFS ("ZERO-KNOWLEDGE") D'IDENTIFICATION. APPLICATION A LA CARTE A MICROPROCESSEUR.

YVAN STAELENS

Résumé

Ce mémoire a pour principal objectif de présenter les nouveaux protocoles cryptographiques interactifs, mieux connus sous le nom de protocoles "zero-knowledge", et de percevoir leur apport en ce qui concerne l'identification. Ces protocoles sont d'abord décrits à partir de la théorie de la complexité. Le mémoire montre ensuite comment traduire ces protocoles en vue d'applications nouvelles utilisant, notamment, la carte à microprocesseur.

Les principaux mots-clés sont : "zero-knowledge", sécurité inconditionnelle, systèmes d'authentification à partir de clés publiques, transfert équivoque, canal subliminal, blob.

Abstract:

The main goal of this dissertation is to explain the new interactive cryptographic protocols, better known as zero-knowledge, and to show their impact about identification systems. Those protocols have a theoretical background inspired by the theory of complexity. The dissertation describes how to translate such protocols to obtain applied ones, especially on smart cards.

The main key-words are : zero-knowledge, unconditionnal security, authentication systems based on public-keys, oblivious transfer, subliminal channel, bit commitment schemes.

Mémoire de licence et maîtrise en Informatique

Juin 1988

Promoteur **Mr J. RAMAEKERS**

Je remercie Mr J. Ramaekers d'avoir accepté la direction de ce mémoire.

Je remercie tout particulièrement Mr J.J. Quisquater, de Philips Research Laboratory à Bruxelles, qui a supervisé mon travail avec patience depuis les explications précieuses des articles, tous récents, faisant partie de sa propre recherche jusqu'aux idées et précisions nombreuses qui ont été l'essence même de ce mémoire.

Je remercie Mr G. Brassard, professeur à l'université de Montréal, pour les explications concernant ses articles et pour ses conférences très instructives dont je me suis fortement inspiré.

Je remercie Marie-Elise, ainsi que toutes les personnes qui m'ont aidé et encouragé tout au long de ce mémoire.

APRÈS UNE PREMIÈRE ANALYSE, JE DOIS AVOUER ÊTRE QUELQUE PEU DÉSEMPARÉ PAR LE SYSTÈME CRYPTOGRAPHIQUE EMPLOYÉ DANS CES DOCUMENTS ! IL NE S'AGIT PAS DES CLASSIQUES TRADUCTIONS STYLE MANCHOU ANCIEN, NI DE GRILLES DE LETTRES OU DE CHIFFRES. JE PENSE QU'IL S'AGIT D'UN SYSTÈME ALÉATOIRE ! CE QUI RENDRA MA TÂCHE EXCESSIVEMENT LENTE ET DIFFICILE ...

LISTE DES ABREVIATIONS.

BP : Blinding Property.

BCS : Bit Commitment Scheme.

DES : Data Encryption Standard.

DLS : Discrete Logarithm Scheme.

ID : IDentification number (public).

NP : classe Non déterministe en temps Polynomial.

OT : Oblivious Transfer.

P : classe Polynomiale.

PIN : Personal Identification Number.

QRS : Quadratic Residuosity Scheme.

RSA : méthode de Rivest-Shamir-Adleman.

SID : Secret IDentification number.

PRESENTATION

* * * * * * * * * * * * *

CHAPITRE I

INTRODUCTION.

Les exigences de contrôle d'accès fiables face aux piratages de plus en plus nombreux liés à la fragilité des supports informatiques (altération de bases de données, falsification de cartes magnétiques, décodeurs illégaux pour chaînes de télévision à péage,...) ont, depuis 1979, incité les spécialistes en cryptographie à concevoir des protocoles sécuritaires au départ d'une approche méthodologique nouvelle proposée par A. Shamir [Shamir, 79].

Le principe original est : tout fraudeur potentiel ne doit rien pouvoir déduire d'un protocole d'identification. Pour ce faire, le protocole ne doit apporter aucune information autre que l'identification a réussi ou l'identification a échoué, c'est-à-dire un seul bit.

La méthodologie a abouti à la construction paradoxale de protocoles à apport nul de connaissance, mieux connus sous le nom de zero-knowledge, et dont il est possible de démontrer de façon théorique que leur sécurité est inconditionnelle.

Ce mémoire fait un état de l'art en matière de ces nouveaux protocoles cryptographiques d'identification, au centre d'une recherche particulièrement active.

Dans une première partie (chapitre II au chapitre IV), le mémoire montre comment, depuis les chiffres à clés publiques, les idées des premiers protocoles zero-knowledge ont évolué vers une formulation théorique inspirée à la fois de la théorie de l'information et de la théorie de la complexité.

Le chapitre II met en évidence le lien entre le concept d'aléatoire et celui d'apport nul de connaissance. Le chapitre III situe le contexte cryptographique opérationnel à partir duquel sont nés les protocoles zero-knowledge, c'est-à-dire les schémas de chiffrement/déchiffrement et de signature propres aux algorithmes classiques à clés secrètes (DES) ou publiques (RSA).

Le chapitre IV est central : à partir de problèmes non-déterministes en temps polynomial, il décrit de façon constructive le principe général d'un zero-knowledge et illustre les rouages subtils de quelques exemples fondamentaux. Le concept élémentaire de BLOB, introduit par G. Brassard, se greffe ensuite logiquement pour décomposer en niveaux des aspects très différents du protocole. Enfin, des applications théoriques dérivées, et dont l'avenir est prometteur, sont présentées.

La seconde partie de ce mémoire (chapitre V au chapitre VII), s'occupe des problèmes et limites induites par les applications pratiques, surtout en ce qui concerne la carte à microprocesseur.

Le chapitre V dérive les protocoles théoriques vers des applications bancaires (par exemple) et montre les directions à prendre si le contrôle d'une autorité est envisagé ou non. Des difficultés d'ordre pratique surgissent : le problème du canal subliminal en est un bon exemple. Entre autres solutions, ce mémoire présente un protocole d'identification en une passe, quasi opérationnel, développé par L. Guillou (du CCETT*, Rennes) et J.J. Quisquater, qui réduit au maximum la possibilité de fraude. Le chapitre VI s'intéresse aux différentes attaques : dans un premier temps sont exposées les attaques mathématiques, tributaires de la puissance des algorithmes probabilistes; ensuite, les attaques liées à l'implémentation et aux contraintes du monde réel, avec un tour rapide des fraudes et des remèdes appropriés. Le chapitre VII concerne l'implémentation proprement dite d'un zero-knowledge simple, qui illustre les difficultés au niveau des techniques de manipulations de grands entiers, de l'interactivité et des générateurs aléatoires.

Le programme implémenté se trouve en annexe.

(*) Centre Commun d'Etudes de Télédiffusion et Télécommunications.

Ce mémoire agence, d'une manière progressive et personnelle, les différents concepts issus directement de la recherche actuelle en cryptographie, en essayant de déceler le fil qui, depuis les idées primitives de zero-knowledge, conduit à des applications de premier plan, notamment dans le domaine de l'identification de cartes à microprocesseur. De plus, les protocoles zero-knowledge sont comparés aux protocoles préexistants, et leur apport spécifique est mis en évidence. Le mémoire présente au lecteur, dans la mesure du possible, une vision structurée des axes de recherche liés directement ou indirectement aux protocoles interactifs d'identification. Ce point est rendu particulièrement difficile suite au manque de recul étant donné la jeunesse de ces protocoles. Ce mémoire tente également de familiariser le lecteur à la manipulation de grands nombres entiers par la présentation d'un programme démonstratif.

On ne peut aborder ce mémoire sans citer une autre nouvelle direction de la cryptographie, presque totalement différente, et qui ne sera pas développée. Il s'agit des travaux originaux, encore théoriques aujourd'hui, de C.H. Bennett et G. Brassard se rapportant à la cryptographie quantique, basée sur la polarisation de rayons lumineux [Bennett & Brassard, 84]. L'avantage essentiel de ces travaux réside dans le fait qu'il n'est pas nécessaire de faire appel à la théorie de la complexité, surtout en ce qui concerne l'aspect "intraitable" de problèmes tels que la factorisation de grands entiers.

PREMIERE PARTIE
****

PROTOCOLES THEORIQUES.

CHAPITRE II

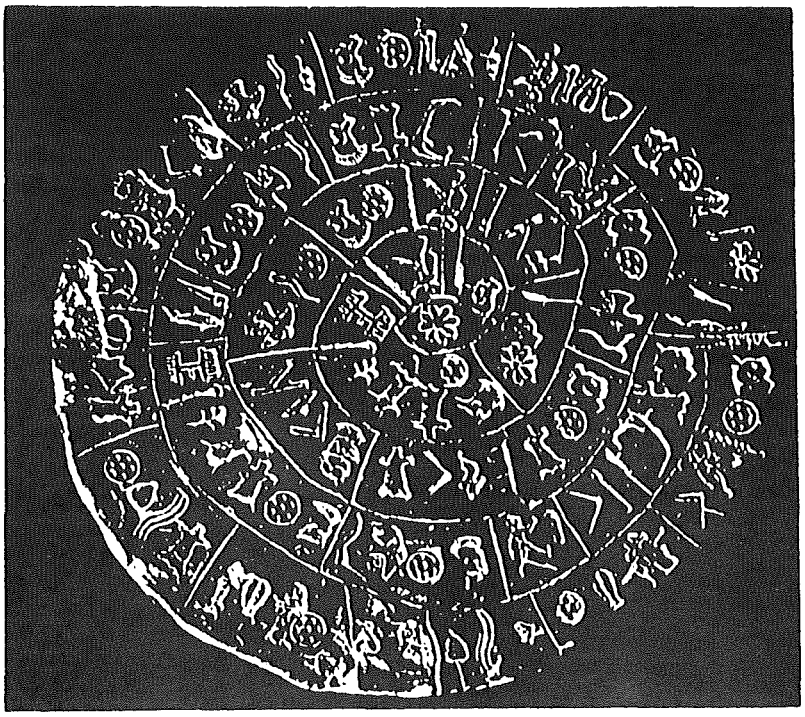
ALEATOIRE ET CRYPTOGRAPHIE.

II.1 LE CHIFFREMENT IDEAL UTILISE L'ALEATOIRE.

Par définition, un message bien chiffré est un message dont il est difficile de retrouver le sens ou plus exactement l'information. Un message d'information est un message dont le degré d'organisation est élevé. La complexité relève du langage et de sa grammaire qui structurent l'information. Chiffrer un message a pour but de faire perdre l'aspect de son organisation. Une méthode de chiffrement apporte donc un désordre. L'esprit humain ne peut plus coordonner le message et il se produit véritablement une crise [Atlan, 79]. On relève un paradoxe : toute organisation tend vers un désordre qui est caractéristique de l'entropie de l'univers et qui, par la présence de ce qu'il faut bien appeler hasard, dégrade les structures informationnelles ([Ekeland, 84], [Prigogine, 82]); tandis que les méthodes cryptographiques éprouvées exigent des structures très élaborées pour masquer ou plus exactement "casser" l'information. La différence vient du fait que l'entropie est irréversible alors que toute méthode cryptographique doit être **reproductible** par un processus de chiffrement/déchiffrement.

Le hasard, encore appelé bruit, est la négation de l'information. A l'extrême, on peut dire qu'un message aléatoire est un message chiffré dont le déchiffrement est inconnu.

On considère, par exemple, l'écriture sous forme d'hiéroglyphes dont on a retrouvé de nombreux exemplaires en Crète. Cette écriture, malheureusement pour les linguistes, n'est pas dérivée des langues indo-européennes et on ne possède pas assez de redondance pour en reconstruire la signification. L'alignement des caractères nous apparaît donc aléatoire [Fig II.1]. Cette organisation nous échappe et c'est la raison pour laquelle, actuellement, cette écriture reste indéchiffrée. Ces ancêtres possèdent donc une méthode de chiffrement efficace à nos yeux parce que la structure qu'ils emploient est en porte-à-faux avec toutes les structures que nous connaissons. Cependant, il suffirait d'un Champollion pour faire tomber la barrière.



Le fameux «disque de Phaistos» en terre cuite. Les signes ne sont pas incisés, mais imprimés dans l'argile humide au moyen de poinçons annonçant les «caractères mobiles». L'inscription, en hiéroglyphique, est vraisemblablement d'origine anatolienne et remonte à 1600 avant J.-C. Cette écriture n'a pas été déchiffrée, bien qu'elle ne comporte que 45 caractères différents. Elle n'a probablement aucun rapport avec la civilisation crétoise (Musée d'Héraklion).

FIG II.1 ECRITURE INDECHIFFREE.

Dans ce cas, l'évaluation de la complexité apparaît difficile. De plus, il n'y a pas de complexité dynamique, à savoir dépendant d'un paramètre.

La cryptographie recherche des méthodes indépendantes des structures de langages. Cela est malheureusement impossible. Néanmoins, les mathématiques et en particulier la théorie des nombres offrent des structurations indépendantes de toute logique humaine. Une faculté étonnante induite par ces structurations abstraites est que l'homme ne peut modéliser entièrement les propriétés des nombres et en particulier des nombres premiers. Même si on trouve de nouveaux théorèmes qui sont des pas de géant faisant ainsi reculer les limites des connaissances, ces pas seront toujours finis dans cet univers infini. En fait, la complexité augmente avec la grandeur des nombres.

	8473	8474	8475	8476	8477	8478	8479
211	388600	125442	482038	713725	472600	893438	014498
212	663764	013579	341495	635933	298302	696122	930616
213	167980	070401	219323	259738	749030	822633	827194
214	318731	588206	794011	576705	635243	401132	897147
215	653663	542372	357618	234676	046456	854896	038732
216	580745	138460	864060	147589	814193	931487	414204
217	736025	329981	032991	985614	031395	281035	095420
218	445837	306526	603468	206282	139937	655224	079933
219	641110	545473	478238	475948	266111	083507	763416
220	217374	416776	135951	203826	861282	984365	114305
221	841199	819851	204309	658535	211631	588335	208568
222	600286	535473	548202	837537	026485	988909	478128
223	309740	675331	090932	084295	517153	113030	116230
224	471628	852168	661790	496558	980492	284626	530915
225	171976	394389	765586	897144	771971	100249	704802
226	177417	109200	451646	321812	109415	925455	117750
227	279792	376525	764909	724871	526714	140872	419216
228	717906	727080	713969	111044	030483	333114	592882
229	920932	380657	990125	224941	408777	188500	131937
230	189487	742243	452022	632318	261534	130166	712829

Part of a table of powers and exponentials, modulo 999 983

FIG II.2 EXTRAIT DE TABLE MODULO.

De plus, le comportement des espaces modulo [fig II.2] met en évidence un caractère pseudo-aléatoire qui remet en cause la notion d'ordre originel en mathématiques. Ceci est en contradiction avec une optique déterministe de la science des nombres. On peut, à ce sujet, se demander si ce n'est pas l'homme qui adapte la structure de la théorie des nombres à son mode de pensée; ceci permet de laisser une place au hasard dans un monde où nombreux sont ceux qui l'auraient banni. Dans ce cas, on parle bien de construction des mathématiques et non pas de redécouverte au sens de Leibnitz ou des platoniciens.

Le déterminisme situe le hasard dans un manque d'explication lié à la limite de l'état des connaissances. Le hasard ne dépend donc que de l'état des connaissances. Si le monde est réellement déterministe, alors l'utilisation de l'aléatoire en cryptographie ne permettra jamais de construire un système inviolable ou plus exactement éternel car il sera toujours possible, un jour, de montrer que l'aléatoire n'en était pas un parce qu'on a réussi à le reproduire! Et ce qui est reproductible pour celui qui chiffre et déchiffre le sera aussi pour tout ennemi.

Mais l'informatique est déterministe parce qu'elle travaille sur des bits : elle est limitée à des nombres finis qui en font un outil abstrait qui modélise difficilement le monde physique.

On est contraint à l'utilisation de méthodes de complexification. Si on complexifie par l'utilisation de la théorie des nombres à un point tel qu'on ne maîtrise pas encore la façon dont se structure l'information, le message ainsi transformé nous semblera aléatoire.

L'aléatoire doit cependant s'immiscer de façon judicieuse. En effet, en téléinformatique, le moindre bruit est très facilement destructeur, la complexité étant fragile par manque de redondance; la fragilité est due surtout à l'unicité d'interprétation de protocoles au bit près. En génétique, par contre, une perturbation aléatoire peut même enrichir l'information (par exemple une mutation); dans ce cas, l'aléatoire ne s'immisce pas au hasard puisque plusieurs interprétations cohérentes sont possibles.

II.2 COMMENT "CASSER" L'INFORMATION DE LA MANIERE LA PLUS EFFICACE POSSIBLE?

On peut appliquer une méthode très simple de chiffrement dite de VERNAM. Cette méthode, utilisée dans certains cas par l'armée, répond idéalement aux exigences développées jusqu'à présent. Il suffit de mêler un message informatique bit à bit à une bande aléatoire de même longueur que le message, en utilisant le "OU EXCLUSIF" [Fig II.3]. De ce fait, le résultat obtenu, ou chiffrement, est aussi une bande aléatoire (il n'y a donc pas de méthode plus efficace pour "casser" l'information). Pour déchiffrer, le principe est identique : la bande chiffrée est mêlée à la même bande aléatoire, ce qui par propriété du "OU EXCLUSIF" redonne le message en clair. Toutefois, cette méthode est souvent irréalisable pour deux raisons :

1°) il faut autant de bandes aléatoires que de messages et chaque bande doit être aussi longue que le message.

2°) aussi bien celui qui chiffre que celui qui déchiffre possède la bande aléatoire.

Le point 2 expose le problème, central en cryptographie, de **transmission** de bande. Ceci est à relier avec le problème très général de transmission de clés. Car en fait, chaque bande aléatoire est une clé.

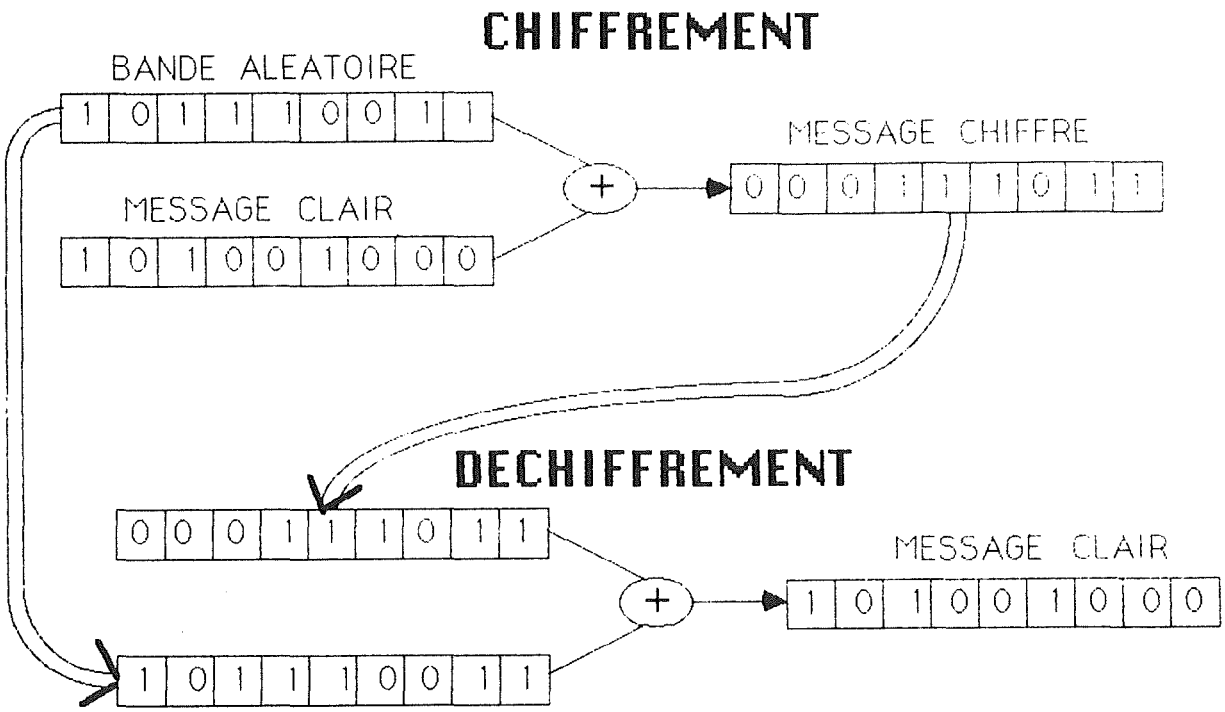


FIG II.3 LA METHODE VERNAM.

II.3 LES GENERATEURS ALEATOIRES.

Le principe même des générateurs aléatoires d'ordinateurs (même non biaisés), est généralement pseudo-aléatoire parce que **reproductible**. Il faut donc distinguer un bon générateur pour des applications statistiques, d'un bon générateur cryptographique. Si on considère les plus élémentaires, ils se basent sur des fonctions réentrantes (registres à décalage, DES à message constant et dont la sortie devient la clé,...) [cfr VII.3]. Tous les espoirs de ces méthodes sont basés sur des périodes très longues avant cyclage. La semence (ou première valeur injectée) détermine malheureusement une suite qui n'a rien d'aléatoire. On ne fait que déplacer le problème en reportant l'aléatoire sur un bon choix de semence. Néanmoins, pour ces méthodes, l'idéal est de ne pas cycler. Malheureusement, le pourcentage moyen de petits cycles, pour une fonction aléatoire quelconque, est alarmant [Quisquater, CRYPTO'86].

Les meilleurs générateurs cryptographiques, en théorie du moins, sont ceux basés sur des appareils de détection de bruits physiques liés à des collisions en physique nucléaire. Quand on sait que la mécanique ne peut résoudre un problème à trois corps, que dire des chocs entre n particules? Dans cette optique, des théories sur le mouvement brownien se basent sur des chocs aléatoires et la seule façon de modéliser est le calcul des probabilités et la statistique. Les générateurs de ce type sont inopérants car très encombrants et surtout hors de prix.

De façon pratique, les générateurs sont souvent basés sur l'horloge interne de l'ordinateur. Mais il est de l'intérêt de l'utilisateur de se rendre compte que le nombre de bits effectivement aléatoires ainsi extraits est faible (de l'ordre de 19 bits pour les horloges précises jusqu'à la microseconde).

En fait, il existe une inertie au hasard liée, pour les méthodes de permutations par exemple, à des lois de groupe qui réduisent la taille des cycles [cfr VII.3].

Ce mémoire se limite au niveau software de protection. Il suppose donc l'existence de générateurs aléatoires tout comme celle de zones de calcul et de mémoire protégées (vis-à-vis des ultraviolets par exemple).

II.4 DEFINITION D'UN MODELE CRYPTOGRAPHIQUE.

La cryptographie, exactement comme les autres sciences, doit définir le modèle théorique sur lequel elle se base. Mais la cryptographie est aussi un art car le modèle théorique est souvent éloigné de la réalité. De ce fait il est nécessaire de classer les différentes méthodes en fonction du niveau et de la qualité de protection exigés. Une méthode répond à un problème précis et à une attaque possible. Généralement, une protection efficace devra se baser sur la combinaison de méthodes élémentaires. Car une trop grande concentration sur une attaque précise permet des fraudes d'un type tout autre.

Des systèmes entiers très élaborés peuvent ainsi s'effondrer suite à des attaques détournées dont le but est parfois tout autre que l'aspect premier vers lequel la protection était dirigée. Il s'agit donc d'être très prudent d'autant plus que bien souvent l'édifice ne tient qu'à un fil, c'est-à-dire une clé. Ceci est particulièrement vrai dans la réalisation pratique des algorithmes où les hypothèses sont bien différentes de celles des considérations théoriques. Les hypothèses sont donc essentielles. Il faut essayer de les définir à chaque fois de façon précise.

CHAPITRE III

CHIFFRES A CLES PUBLIQUES.

III.1 CATEGORIES DE CRYPTOSYSTEMES.

Historiquement on subdivise les cryptosystèmes en trois catégories dépendant des types d'algorithmes et de clés. Ce mémoire étudie par la suite une nouvelle catégorie basée sur des méthodes dites de "zero-knowledge" et dérivée de la troisième catégorie. C'est la raison pour laquelle celle-ci sera particulièrement développée.

La première catégorie, la plus ancienne, est celle des algorithmes secrets à clé(s) secrète(s).

La seconde catégorie, d'utilisation courante, est celle des algorithmes publics à clé(s) secrète(s).

La troisième catégorie est celle des algorithmes publics à clé(s) révélée(s).

Problème général relatif à tous les algorithmes :

- L'algorithme est valable, mais jusqu'à quand? Si quelqu'un le "casse", c'est très grave. Un algorithme "cassé" n'est pas réparable.

- La structure des algorithmes entraine une complexité de réalisation d'une forte hiérarchisation de clés. En effet, leur élaboration est rarement pensée en terme d'organisation et de gestion de priorités.

Problème général relatif aux algorithmes à clés secrètes :

- Des clés secrètes doivent être changées. Il y a risque d'interception lors d'un transport de clés.

III.1.1 Algorithmes secrets à clé(s) secrète(s).

L'algorithme ainsi que la ou les clés doivent être conservés secrets. L'algorithme Télépass de la carte CP8 de Bull est de ce type [De Pra, 87].

Avantage:

- Le secret de l'algorithme empêche les attaques basées sur la façon dont il a été construit et sur son comportement attendu.

Désavantages:

- Un algorithme secret est généralement développé en petit comité. Sa résistance aux attaques a été étudiée en petit comité également. La probabilité d'une faille est grande, parce que l'algorithme est peu éprouvé.

- L'algorithme est soumis à des restrictions d'usage : il ne peut être transmis, par exemple d'une manière software.

III.1.2 Algorithmes publics à clé(s) secrète(s).

L'algorithme est publié mais les clés sont toujours secrètes. L'algorithme Data Encryption Standard (DES) en est un bon exemple [Davies & Price, 80].

Avantages:

- L'algorithme a été éprouvé par des études aussi bien théoriques que pratiques émanant de divers centres de recherche. La probabilité d'une faille est faible.

- L'algorithme est standard et s'adapte très bien pour des firmes commerciales qui ne développent pas leur propre algorithme.

- L'algorithme est adapté aux grands réseaux.

Désavantage:

- Comme l'algorithme est très utilisé, l'intérêt de le "casser" est grand, et donc, peut-être, les moyens mis en oeuvre pour ce faire seront grands.

III.1.3 Algorithmes publics à clé(s) révélée(s).

Le principe de ces méthodes relativement récentes a été inventé par Diffie et Hellman [Diffie & Hellman, 76]. Ce principe se base sur un chiffre asymétrique contrairement au DES. En fait, seule la clé de déchiffrement reste secrète. La clé de chiffrement fait partie d'un catalogue public qui la lie au correspondant. La méthode Rivest-Shamir-Adleman (RSA) prend place dans cette catégorie [cfr III.4.2].

Avantages:

- Le problème de distribution de clés est simplifié surtout dans le cas d'une forte hiérarchisation de clés : dans ce cas, il y a plusieurs niveaux d'utilisateurs classés par priorités. Ceci est rendu possible par une diminution des endroits où se trouve la clé secrète (par exemple auprès de l'autorité uniquement).

- Les calculs sont relativement simplifiés (au moins pour la vérification de signature) et s'implémentent sur une carte à puce par exemple.

- La sécurité de l'algorithme peut être évaluée dans le cas où l'algorithme fait appel à la théorie de la complexité (factorisation dans le cas du RSA).

- Les méthodes du type RSA permettent des schémas simples et distincts de chiffrement (confidentialité) et de signature (intégrité) [cfr III.4.2].

Désavantages:

- L'authentification de la clé publique est à distinguer de l'identification du correspondant : autrement dit, la clé publique utilisée est-elle bien celle du récepteur voulu?

- Il n'y a pas d'identification certifiée de l'émetteur.

- Il y a danger dans le cas d'appropriation de la clé publique par l'ennemi pour des modifications partielles liées à la division en bloc du message (s'il n'existe pas d'empreinte) [cfr attaque date d'anniversaire III.3.4].

- Il existe toujours des problèmes de changements de clés publiques, puisqu'une clé, au moins, est toujours secrète. Par conséquent, il faut également engendrer de nouvelles clés secrètes.

III.2 IDENTIFICATION ET AUTHENTIFICATION.

L'authentification d'un message consiste à être convaincu en temps, en lieu, en nature, de l'exactitude du message. L'authentification de message n'est pas seulement utile pour vérifier qu'un ennemi n'a pas la possibilité d'envoyer un faux message. En effet, généralement celui qui envoie un message et celui qui le reçoit sont en concurrence. Celui qui envoie peut mentir en disant que ce n'est pas lui qui l'a envoyé. C'est pourquoi, dans bien des applications, un arbitrage est conseillé.

Avec et sans arbitrage, des signatures sont nécessaires pour authentifier des messages. Le principe d'une signature réside dans le fait qu'elle doit être inimitable. La signature du message fait office de preuve tandis que le récepteur fait office de vérificateur. Le message peut passer par plusieurs intermédiaires.

Mais si on veut identifier un correspondant, on ne peut tolérer d'intermédiaires que sous d'énormes précautions. En effet, le certificat d'identité, unique, ne peut se communiquer. On confond souvent signature et identification, surtout dans le cas d'accréditations qui sont en fait des certificats et donc des signatures. Contrairement à l'authentification, il ne peut pas avoir plusieurs accréditations pour l'identification.

Ceci ne peut se résoudre par aucun des trois types d'algorithmes cités précédement, comme on va le montrer. Par contre, si on se munit d'énormes précautions, les procédures interactives du type zero-knowledge permettent de résoudre des problèmes d'identification sous certaines hypothèses.

III.3 LES FONCTIONS A SENS UNIQUE.

L'usage de fonctions à sens unique est essentiel en cryptographie car elles sont à la base de nombreux systèmes. Elles dérivent généralement du fait que "inverser" la fonction est un problème de complexité non polynomiale.

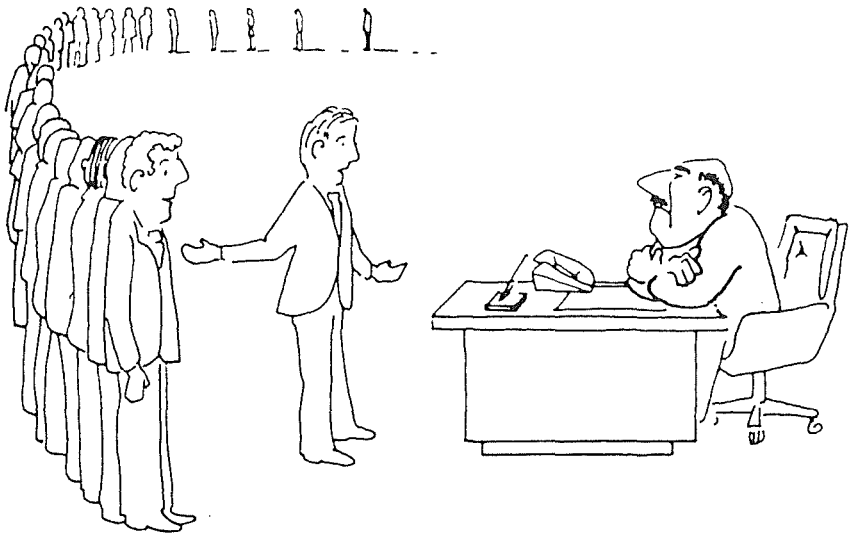
La fonction f est à sens unique si et seulement si étant donnés x et y, vérifier que f(x) = y est de complexité polynomiale ($o(n^k)$) alors que si seul y est donné, retrouver x tel que f(x) = y est de complexité non polynomiale ($o(k^n)$) où k est constant.

Une solidité "perpétuelle" pour un algorithme n'existe pas. Mais on peut néanmoins la réaliser relativement à un paramètre qui fait varier de façon conséquente la difficulté (ici n, exponentiel). Généralement ce paramètre est déterminé par la grandeur de nombres entiers. Par contre, le DES travaille sur des clés fixes (56 bits effectifs). La longueur des clés est donc difficilement modifiable ; on ne détermine pas la solidité du DES directement par la théorie de la complexité. Si on "casse" le DES par une attaque mathématique (i.e. non exhaustive), c'est son principe même qui est mis en péril.

III.3.1 Les nombres premiers engendrent des problèmes "intraitables" facilement implémentables:

Soient p, q deux nombres premiers très grands. On définit N = p*q. La factorisation est bien sûr unique, à l'ordre près (espaces modulo). Retrouver p et q à partir de N peut se réduire à un problème dans NP\P [IV.1.1]. La complexité est non polynomiale. Donc, pour se fixer les idées, si les performances des ordinateurs se multiplient par 1000 et si les méthodes de factorisation sont 10 fois meilleures, la complexité est augmentée de l'ordre de Log(1000x10) = ±5 fois plus grand (donc un chiffre en plus seulement).

Les méthodes les meilleures sont celles qui se ramènent à des problèmes de complexité **étudiée** et donc standard . Plutôt que montrer qu'il est difficile de résoudre ce problème parce qu'on n'y arrive pas, on montre que c'est difficile parce que personne n'y arrive [Fig III.1]. En fait, on montre que les algorithmes permettant de résoudre ce problème sont tous inefficaces parce qu'ils sont équivalents à essayer toutes les possibilités, sans stratégie et donc de façon combinatoire.



"I can't find an efficient algorithm, but neither can all these famous people."

FIG III.1 AVANTAGE DU PRINCIPE DE REDUCTION [GAREY & JOHNSON, 79].

III.3.2 Exemples.

Exemple 1: Système Diffie-Hellman (schéma exponentiel).

Si p est premier, $M' = f(M) = g^m \pmod p$ est facile à calculer, mais la découverte de f^{-1} est impossible, en un temps raisonnable [fig III.2], si g et p sont suffisamment grands [Davies & Price, 88].

Nombre de bits de p	Temps de calcul SEQUENTIEL pour trouver M
200	4 heures
300	2 ans
400	3.8 milliards d'années
500	48 millions de milliards d'années

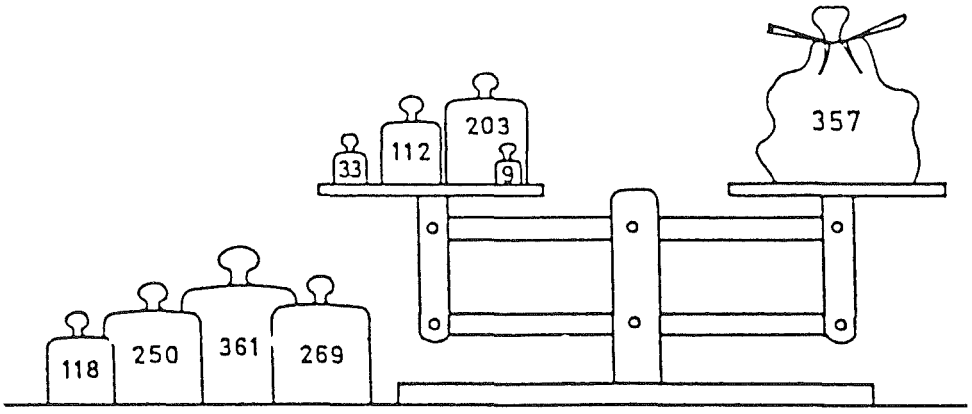
[FIG III.2] ORDRE DE TEMPS DE FACTORISATION D'UN NOMBRE P.

Exemple 2: Système Knapsack (ou empilements, ou poids).

Suite aux travaux de Ralf Merkle et de Martin Hellman (USA) dans les années 70, on a utilisé un problème qui est d'ailleurs de type NP-Complet [cfr IV.1.2].

Ayant un ensemble de nombres $(a_1, a_2 \ldots, a_n)$, il faut trouver tous les sous-ensembles tels que la somme des nombres contenus dans chaque sous-ensemble soit égale à un nombre donné H (H=357 dans l'exemple [Fig III.3]).

Malheureusement, ce système est "cassé", malgré la NP-Complétude du problème utilisé : on n'est jamais assez prudent en cryptographie.



The knapsack problem

FIG III.3 KNAPSACK.

III.3.3 Types de Fonctions.

On considère un ensemble de fonctions paramétrées par la clé k et calculables efficacement : x --> y = f(x,k).

Le "sens unique" sur la clé est la difficulté de recherche de la clé k à partir du couple (x,y).

Le "sens unique" sur l'argument est la difficulté de recherche de x à partir de y et de k.

Une fonction de **compression** est de type injectif. La propriété essentielle des fonctions de compression est la difficulté de mettre en **collision**, c'est à dire de produire une paire d'arguments telle que f(x',k) = f(x'',k). Les collisions sont gênantes pour l'intégrité du message si elles sont nombreuses : il n'y a pas unicité, d'où il n'y a pas empreinte. Dans la méthode Knapsack, par exemple, plusieurs sous-ensembles peuvent satisfaire comme solution. Il est important de mettre en oeuvre des méthodes de diversification (**anti-collision**).

La fonction f possède un **inverse à gauche** si et seulement si il existe g telle que pour tout x, x = g(f(x,k),k). Si f = g alors le système est symétrique et la fonction f est dite **involutive**.

Une fonction f est dite **à trappe** si il existe une trappe dont la valeur permet le calcul efficace de l'inverse à gauche de la fonction f.

Une paire de fonctions fa et fb est dite **commutative** si et seulement si fa(fb(x,k),k) = fb(fa(x,k),k). Les paires commutatives se mettent facilement en collision. La connaissance de la valeur de la trappe est équivalente à la mise en collision pour des paires anti-collision [Fig III.4].

NOTE : La phrase "g peut être déduite de f" signifie que, à partir de l'algorithme f gouverné par la clé k, on sait bâtir un algorithme efficace pour réaliser son inverse à gauche. Le symbole (*) indique que la classe de fonctions dans la colonne possède la propriété indiquée dans la ligne.

y = f(x, k)	Systèmes symétriques	Fonctions à sens unique			Systèmes Asymétriques
		Fonctions de Compression	Paires Commutatives	Paires anti-collision	
k est difficile à déduire de (x, y)	*				*
x est difficile à déduire de (y, k)		*	*	*	
f a un inverse à gauche g déduit :	*			*	*
facilement	*				
par une trappe				*	*
Commutativité			*		
Difficulté à mettre en collision				*	

FIG III.4 TYPES DE FONCTIONS (CRYPTOLOGIA 88. A PARAITRE).

III.3.4 Applications élémentaires.

Les fonctions à sens unique permettent de protéger la table des mots de passe dans un ordinateur. On enregistre alors leurs images : f(mot de passe).

Méthode de Lamport.

Les fonctions à sens unique permettent un procédé d'identification à mots de passe variables utilisés une seule fois. Comme pour les générateurs aléatoires - du moins, un principe voisin - l'utilisateur tire au hasard (!) une semence qu'il garde secrète. Il donne à l'ordinateur la valeur f^{10000}(semence) obtenue en appliquant 10000 fois la fonction à sens unique. L'utilisateur donne $f^{10000-K}$(semence) comme mot de passe au Kième accès.

Les fonctions de compression donnent ce qu'on appelle une **empreinte** du message. Elles évitent, si l'empreinte est suffisamment grande, les attaques du type "date d'anniversaire" liées souvent à la décomposition en bloc du message. En effet, lorsque dans une assistance quelconque on regarde s'il y a des personnes nées le même jour, on s'aperçoit bien souvent que c'est le cas. Ce résultat surprenant est bien connu des probabilistes : la probabilité qu'effectivement il y en ait est non négligeable (proche de 2/3 (en fait :1-1/e)). Ce problème surgit dans le choix des méthodes de hashing en base de données. De la même manière, en cryptographie, le risque de collision est non négligeable. Si le message est découpé en blocs, un ennemi peut dévier le sens des messages en modifiant les parties où il a repéré des collisions (après avoir préalablement testé un nombre modéré de clés). En quelque sorte, l'empreinte n'est pas unique. Les fonctions de compression se basent souvent sur le principe de rendre dépendants x et k dans f. Une idée effectivement appliquée consiste à définir x' = x <ou exclusif> k et de calculer ensuite y = f(x',k). Enfin, les injections sont utiles également pour éviter les attaques par canal subliminal ("subliminal channel") [cfr V.2].

III.4 CHIFFRES A CLES PUBLIQUES.

III.4.1 Schémas à une passe (type RSA déterministes).

Le système RSA doit son sigle aux noms de ceux qui le publièrent en 1977 : Ronald RIVEST, Adi SHAMIR, Leonard ADLEMAN. Leur système se fonde sur les fonctions à sens unique [Rivest, Shamir & Adleman, 78].

Chaque membre du réseau, utilisant ce cryptosystème, choisit deux grands nombres premiers p et q et calcule N = p*q. Il choisit encore un nombre R premier avec T = (p-1)*(q-1). La clé publique sera (R,N). Pour chiffrer un message, on le découpe en blocs Mi de nombres ayant le même nombre de chiffres mais de valeur inférieure à N. On calcule $Mi' = Mi^R \pmod N$. La partie secrète s'appuie sur les deux nombres p et q que l'on ne peut trouver à partir de la seule connaissance de N. La clé secrète du destinataire ayant publié (R,N) est $S = R^{-1} \pmod T$. En fait, on calcule S en résolvant l'équation $R*S = 1 \pmod T$. Une fois qu'il a calculé S, il peut même oublier p , q et T. Le déchiffrement se fera par le même algorithme que le chiffrement grâce à la commutativité de l'exponentielle. En effet, $Mi = (Mi')^S \pmod N$. On garde S secret et R , N sont publiés.

Démonstration.

Le principe étant le même pour tout bloc i, on a:

$$(M')^S = (M^R)^S = M^{R*S} \pmod N$$

$$R*S = 1 \pmod T$$

$$R*S = K*T + 1$$

$$R*S = K*(p-1)*(q-1) + 1$$

or $$M^P = M \pmod p$$

$$M^{P-1} = 1 \pmod p \quad \text{(Thm de Fermat)}$$

$$(M^{P-1})^{K*(Q-1)} = 1 \pmod{p}$$

$$M^{K*(P-1)*(Q-1)+1} = M \pmod{p}$$

$$M^{R*S} = M \pmod{p}$$

de façon symétrique $M^{R*S} = M \pmod{q}$

d'où $M^{R*S} = M \pmod{p*q}$ (Propr. des congruences)

■

III.4.2 Confidentialité et intégrité par clés publiques.

Les clés publiques permettent des schémas de signature séparés des schémas de chiffrement.

Si on considère un schéma de signature par clés secrètes, on a forcément une table secrète par correspondant dans laquelle est placée l'identifiant du correspondant et la clé secrète qu'il faut utiliser pour correspondre avec lui:

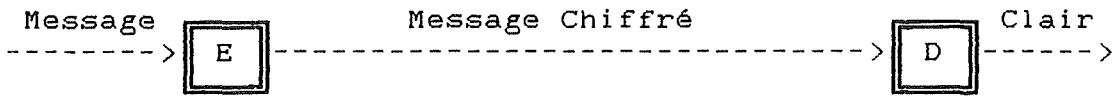
Correspondant A

B	clé e

Correspondant B

A	clé d

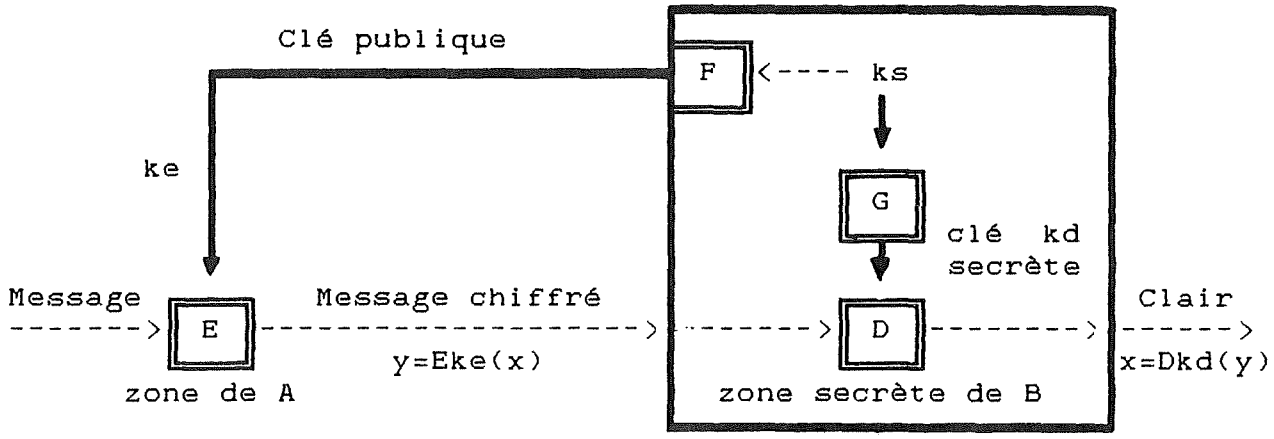
S'il s'agit d'un schéma symétrique comme le DES, alors d = e.



Il y a malheureusement transport de clés ! Et les schémas de signature sont difficilement séparables des schémas de chiffrement.

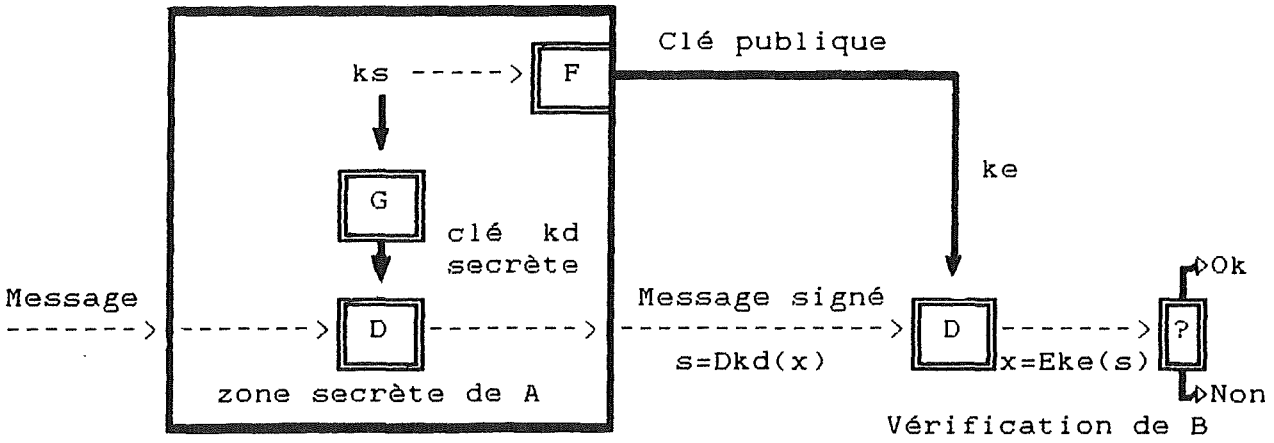
Chiffrement par clé publique:

On applique le principe de la méthode RSA avec des clés distinctes. Seul celui qui reçoit le message (ici B) doit pouvoir le déchiffrer, d'où utilisation de sa clé secrète pour déchiffrer. Mais si A décide de changer de clé, il le fait savoir à B (ou simplement dans le cas où B décide de changer de clé). Celui-ci recalcule une nouvelle clé publique pour A [Davies & Price, 80].



Signature par clé publique:

Comme on l'a déjà fait remarquer, une signature authentifie un document et c'est la raison pour laquelle tout le monde doit pouvoir vérifier si le message est authentique. Pour ce faire, on doit utiliser une propriété qui est propre à l'envoyeur A. La solution est de signer le message par l'emploi de sa clé secrète. Et par le schéma, la clé publique permet à n'importe qui possédant le message signé, de vérifier son authenticité. Le procédé est très simple et l'intégrité se réalise par le schéma MIROIR de la confidentialité.



III.4.3 Schémas interactifs (de type "zero-knowledge" probabiliste).

III.4.3.1 Le problème d'identification d'une personne physique.

Comment peut-on identifier une personne?

Dans la vie de tous les jours, on reconnaît une personne par ses traits caractéristiques. Notre cerveau identifie la personne par une vérification de critères enregistrés comme peu communs. Il arrive cependant que, de loin, on se trompe suite à une ressemblance. Dans ce cas, les critères suffisant d'habitude pour l'identification ne suffisent plus. Mais la probabilité de se tromper est faible. Elle vaut en fait la probabilité de retrouver les mêmes caractéristiques chez une autre personne. Pour les personnes que l'on connait bien, la probabilité de se tromper est quasi nulle étant donné que le nombre de vérifications de caractéristiques est très grand. En fait, notre cerveau vérifie de façon interactive l'une après l'autre -ou en parallèle- les caractéristiques de la personne.

Solutions informatisables.

Bien sûr, une vérification informatique doit représenter les caractéristiques sous forme informatique. Il y a donc transport et stockage d'information de type standard. La complexité du problème se révèle facilement par l'analogie suivante : "il faut identifier une personne qu'on n'a jamais vue". De ce fait, on se renseigne auprès d'une autorité (quelqu'un qui connaît et a vu cette personne) et qui fournit les informations permettant l'identification. Moins les renseignements seront communs, plus on a de chance d'identifier la bonne personne. Du point de vue informatique, il faudra donc que les caractéristiques identifient de façon **unique** la personne.

La première idée qui vient à l'esprit -et qu'on applique pour l'instant- est que la personne possède une carte d'identité unique non falsifiable, ou plus précisément un Secret IDentification number (SID). Le tout est que la personne ne puisse pas se séparer de son SID (d'où secret) , ou du moins n'y verrait que des désavantages très importants. On construit en fait une correspondance biunivoque entre la personne et son SID. On peut donc identifier la personne par son SID.

III.4.3.2 Identification à distance.

On ne conçoit une identification informatique qu'entre deux correspondants qui communiquent via une ligne ou un bus. Ceci est valable aussi bien pour des utilisateurs sur un réseau qu'entre une carte et un lecteur.

Si on peut faire confiance à la ligne, c'est-à-dire si on est certain que personne n'interfère dans la communication, alors on peut construire des schémas simples d'identification dérivés des schémas de signature. En effet, si A est le prouveur et B le vérificateur d'identité, il suffit qu'ils appliquent la procédure de signature par clé publique où on a défini la clé secrète de A comme étant le SID! Le schéma est le suivant :

B génère un message M, le communique à A en demandant de le signer.

A signe M en calculant M'= SID(M), renvoie M' à B.

B vérifie que M' est correct grâce à la clé publique.

On peut évidemment répéter la procédure avec un nombre suffisant de messages différents jusqu'à ce que B soit suffisamment convaincu. Les messages M' sont appelés accréditations ou certificats.

Bien sûr les messages doivent être différents à chaque identification. Pour ce faire, B génère des messages aléatoires. Néanmoins, au bout d'un temps relativement court, B connaîtra suffisamment de couples <M,SID(M)> qui lui permettront de construire une table de correspondance qui n'est en fait qu'une fonction partielle de la fonction SID. De plus, la taille des messages M est limitée et donc le domaine sur lequel est définie SID est fini. **Sur la ligne passe de l'information** non négligeable sur SID. En effet, un nombre relativement peu élevé de paires chiffré/clair permet peut-être de façon substantielle d'aider à retrouver SID car, et c'est là le problème essentiel : **aucune** démonstration ne permet d'affirmer qu'il est impossible de retrouver SID. De plus, cette méthode n'est absolument pas efficace car elle exige un annuaire de SID^{-1} aussi grand que le nombre d'utilisateurs.

Au bout d'un certain temps, B -et donc n'importe qui- pourra se faire passer pour A aux yeux d'un autre correspondant de A, soit C. Autrement dit, tout se passe comme si B interférait avec une communication entre A et C. Donc une ligne soit-disant sûre se ramène à une ligne non sûre.

Gageure du problème d'identification à distance.

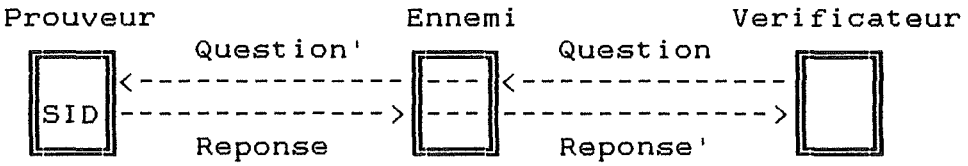
A priori, l'idéal est qu'**aucune information sur le SID ne passe** par la ligne.

Mais en fait, on peut montrer qu'il est théoriquement impossible de construire un système d'identification à distance complètement sûr à cause de l'attaque par transparence.

En effet, si un ennemi s'intercale entre le prouveur A et le vérificateur B autorisé comme tel par A, il suffit à cet ennemi (ce n'est pas si simple quand même !) de se faire passer pour le vérificateur aux yeux du prouveur et de prouveur aux yeux du vérificateur.

Pour chaque question posée par le vérificateur à l'ennemi, l'ennemi la pose sans rien changer au prouveur (et de ce fait, l'ennemi devient un "questionneur autorisé" puisque le vérificateur l'est !). Ensuite, le prouveur répond à l'ennemi en utilisant SID, et l'ennemi reporte la réponse au vérificateur. L'ennemi agit donc de façon transparente et l'ennemi peut ainsi prendre l'identité du prouveur.

On peut chercher à éviter ce problème, en plaçant des hypothèses précises, grâce à des principes de synchronisation et de timing qui ne permettent plus à un ennemi d'allonger le délai entre une question et sa réponse de par le relais.



Temps(Question',Reponse) < Temps(Question,Reponse')

On peut aussi définir des zones neutres qui empêchent une participation d'un ennemi, en se basant sur des transactions dans ces zones gardées [cfr VI.2.2.2 Cages de Faraday et Tamper Free].

III.4.3.3 Procédures à apport nul de connaissance sur la valeur de l'accréditation (ou protocoles "zéro-knowledge").

Pour construire des procédures d'identification (ou authentification du ID) efficaces, on ne peut pas accepter le passage d'informations sur le SID, à savoir la valeur d'accréditation. Les développements qui suivent montrent que de telles constructions sont possibles si on suppose que l'attaque par transparence est impossible.

Depuis 1983, les recherches de Goldwasser, Micali, Rivest [Goldwasser, Micali & Rivest, 84] et Widgerson [Goldwasser, Micali & Widgerson, 86] ont permis de construire des méthodes à apport d'information **nul ou quasi nul** sur le SID. La difficulté de ces nouvelles méthodes est de démontrer qu'elles sont réellement à apport nul. Ceci se fait par l'intermédiaire d'une procédure interactive de démonstration qui consiste généralement à montrer que le vérificateur n'a rien gagné sur le SID ou encore : l'information qu'il possédait avant, lui permettait a priori de déterminer les données recueillies par la procédure interactive. Ceci est paradoxal puisque la procédure a quand même permis de passer l'information réduite à un seul bit "identification OK ou Pas OK" (mais qui est de nature autre qu'une donnée !).

CHAPITRE IV

PROTOCOLES "ZERO-KNOWLEDGE".

IV.1 ELEMENTS DE THEORIE DE LA COMPLEXITE.

IV.1.1 Problèmes de type Non déterministe en temps Polynomial (NP).

Généralement, les problèmes Non déterministes en temps Polynomial (dorénavant appelés NP) sont des problèmes d'existence. Ceux qui nous intéressent sont ceux pour lesquels il est possible de vérifier en un temps **polynomial** si x est une solution ou non. De façon plus formelle :

un problème fait partie de la classe NP si tout système de preuve est facile à VERIFIER mais généralement difficile à TROUVER.

Par exemple, il est difficile de prouver que $2^{67}-1$ est décomposable, mais il est facile de vérifier que $2^{67}-1$ se décompose en 193 707 721 x 761 838 257 287 , dont les facteurs sont d'ailleurs premiers.

La classe des problèmes de type Polynomial (dorénavant appelée P) est la classe des problèmes faciles à TROUVER, dont la complexité est polynomiale, c'est-à-dire de l'ordre d'une puissance de n [$o(n^{cst})$].

Un problème appartient à NP\P si il ne peut pas être résolu par un algorithme en un temps polynomial et si la complexité de ce problème est exponentielle [$o(cst^n)$]. Les problèmes de ce type dérivent généralement de problèmes combinatoires, pour lesquels on ne possède pas de stratégie de résolution plus efficace qu'une recherche exhaustive.

La frontière entre Polynomial et Non déterministe en temps Polynomial n'est pas précise et nombre de problèmes initialement classés dans NP\P passent dans P (notamment en programmation linéaire, concernant le simplexe). Rien ne dit qu'un jour on ne prouvera pas que P = NP. Mais les constantes exposées seront tellement élevées que la complexité sera quasi exponentielle pour les tailles utilisées. L'intérêt de ces problèmes en cryptographie est que la probabilité de trouver la solution dans un temps raisonnable (quelques siècles) est quasi nulle.

IV.1.2 Problèmes NP-Complets.

On essaye de classer les problèmes de NP\P en les réduisant à des problèmes connus. La classe principale est la mieux étudiée, parce que définie mathématiquement, est celle des problèmes NP-Complets.

Lo est NP-Complet si et seulement si

1°) Lo ∈ NP.
2°) Pour tout L ∈ NP, L est **polynomialement réductible à** Lo.

Polynomialement réductible signifie qu'on peut construire un algorithme polynomial qui réduit le problème L à Lo.

Cette définition conduit tout de suite à un corollaire intéressant.

Corollaire 1.

Si A ∈ NP-Complet alors A ∈ P <=> P = NP.

Dém: => A ∈ NP-Complet et A ∈ P.
Soit B ∈ NP: par définition de A, B est polynomialement réductible à A. Or A ∈ P, donc B ∈ P, d'où NP est inclus dans P.
Or P est inclus dans NP, d'où P = NP .

<= évident car A ∈ NP => A ∈ P ■

Corollaire 2 (Méthode de réduction).

Si X ∈ NP, pour prouver que X ∈ NP-Complet, il suffit de prendre un problème Y ∈ NP-Complet (on le sait!) et de montrer que Y est polynomialement réductible à X.

La méthode à l'avantage de ne pas passer par la définition, qui est lourde à cause du "pour tout L".

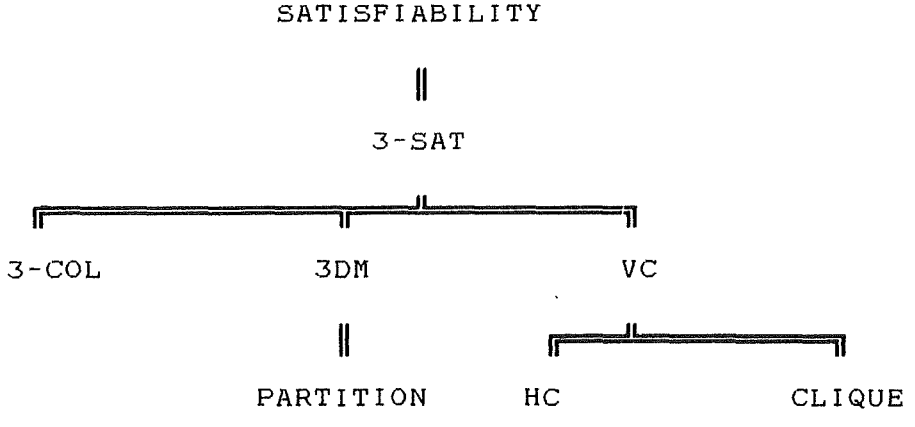
Dém:

Par définition de Y, pour tout L ∈ NP, L est polynomialement réductible à Y qui, par transitivité, est polynomialement réductible à X. Donc X vérifie la définition de la NP-Complétude. ■

La relation de réduction détermine un sous-ordre, que l'on peut représenter sous la forme d'une hiérarchie de réduction dont le sommet est le problème NP-Complet de "satisfaisabilité" SAT. En effet, le corollaire 2 ne fonctionne pas pour le premier problème NP-Complet à considérer à cause de sa formulation récursive. Cook [Cook, 71] a démontré que tout problème NP-Complet peut se ramener par une réduction à SAT. Ceci justifie sa place au sommet.

Le problème de satisfaisabilité consiste à trouver des valeurs (booléennes) qui satisfont une formule (booléenne) quelconque. Il s'agit de formule conforme aux prédicats d'ordre 0.

Hiérarchie de réduction.



où on a défini :

3-SAT : Idem que SAT mais pour les formules à 3 variables. [note : 2-SAT est dans P.]

3-COL : Colorer un graphe avec 3 couleurs seulement.

CLIQUE : Trouver un sous-graphe d'un graphe de telle sorte que ce sous-graphe soit complet.

HC : Trouver un circuit hamiltonien dans un graphe.

PARTITION : Trouver une partition de A, à savoir {A',A\A'}, telle que si on associe un poids à chaque élément x de A, on a :

$$\sum_{x \in A} \text{poids}(x) = \sum_{x \in A \setminus A'} \text{poids}(x).$$

VC : Trouver une couverture transitive ["Vertex Cover"] de taille inférieure ou égale à un nombre donné. [Knapsack se réduit à VC].

3DM : ["3-Dimensional-Matching"]. Trouver un ensemble de coordonnées M' C M C W x X x Y tel que :

M'= # W et il n'existe pas deux éléments de M' ayant une coordonnée commune.

IV.2 "ZERO-KNOWLEDGE" ET NP-COMPLETUDE.

On peut montrer qu'un zero-knowledge peut être associé à chaque problème NP-Complet. Pour ce faire, la technique est de montrer qu'on peut construire un zero-knowledge à partir de SAT, ce qui a été développé récemment à partir de tables booléennes [Brassard, Chaum & Crépeau, 86]. La thèse tient alors par le fait que tout problème NP-Complet peut se ramener à SAT.

IV.2.1 Qu'entend-on par "zero-knowledge" ?

Un zero-knowledge est un protocole interactif qui permet le passage d'un bit [vrai ou faux] et d'un bit seulement. Ceci se définit de façon plus formelle :

> si L est le langage associé au zero-knowledge et si x est une proposition de solution, le protocole vérifie si x appartient à L ou non [Goldwasser, Micali & Rackoff, 85].

Donc l'information qui est passée est bien l'appartenance ou la non-appartenance. En aucun cas l'information n'est x car le principe-même est d'éviter que le vérificateur puisse connaître x ou n'importe quelle information pouvant amener à identifier x.

En résumé, le prouveur montre que x appartient à L et donc que x est une solution effective du problème NP (tous les zero-knowledge se basent sur des problèmes NP), sans que le vérificateur n'ait à aucun moment de l'information sur x.

De façon plus concrète, dans les protocoles d'identification, le prouveur va convaincre le vérificateur que SID identifie ID et cela sans que le vérificateur n'ait aucune information sur SID [SID ϵ L(ID)]. Et de cette manière, le vérificateur ne pourra pas se faire passer ultérieurement pour le prouveur. Il n'a donc acquis aucune information supplémentaire. Donc, ce protocole peut être utilisé de nombreuses fois sans danger puisque aucune information ne passe sur la ligne !

IV.2.2 Mode de réalisation d'un "zero-knowledge".

DEFINITIONS:

Une information est **α-aléatoire** (au moins) si elle n'admet pas de structure fine α (au moins).

Généralement α = (T) Theoreticaly calculable .
= (P) Polynomialy " .
= (I) Infinite Power " .
= (R) Reasonable Power " .

Par structure fine, on entend structure pouvant amener au dévoilement ou à la détermination totale ou partielle de l'information. C'est-à-dire que la probabilité de trouver une structure α pour l'information α-aléatoire est quasi-nulle. Une information sera T-aléatoire si il n'est théoriquement pas possible de la déduire. Une information sera P-aléatoire si il n'est pas possible de la déduire par un calcul en temps polynomial. Une information sera I-aléatoire si il n'est pas possible de la déduire même si on dispose d'une puissance de calcul infinie. Enfin, une information sera R-aléatoire si il n'est pas possible de la déduire en temps raisonable par un calcul. Généralement les applications doivent se limiter aux informations R-aléatoires ou, dans les meilleurs cas, P-aléatoires.

Sous le modèle α, une information **est non-déterministe** pour x si elle se présente sous forme α-aléatoire vis-à-vis de x. Le modèle dépend du problème. Ce mémoire se veut aussi général que possible. Dès lors, le lecteur doit resituer le modèle en fonction du problème.

REALISATION :

Pour réaliser un zero-knowledge, le principe est de partitionner en deux parties une information permettant de retrouver x de telle sorte que A AUCUN MOMENT le vérificateur ne puisse être en possession des deux parties ensemble.

De plus cette partition vérifie les trois propriétés :

- Chacune des deux parties n'apporte **que** de l'information non-déterministe sur x.

- Les deux parties ensemble apportent sans équivoque possible toute l'information x.

- Le vérificateur, ayant en sa possession une des deux parties doit pouvoir déterminer :

    a) c'est réellement une information sur x même si elle est non-déterministe.

    b) il est possible de distinguer de quelle partie il s'agit sans que le prouveur ne puisse tricher.

    c) s'il possédait l'autre partie, l' information qui s'y ajoute (bien que non-déterministe!) permettrait de déterminer x de façon certaine.

On a donc que la somme de deux informations non-déterministes peut être déterministe. Ceci peut paraître paradoxal, mais déjà dans la méthode VERNAM [cfr II.2], ce principe est présent. En effet, la bande aléatoire (au sens T-aléatoire) constitue l'une des deux parties et le message chiffré (tout aussi T-aléatoire) constitue l'autre partie. On vérifie facilement les deux premières propriétés mais pas la troisième. Le problème vient du fait que l'information est de même nature dans les deux parties.

STRATEGIE :

1) Le prouveur décompose l'information comme ci-dessus.

2) Le vérificateur choisit une des deux parties.

On suppose ici que le prouveur ne peut plus changer les parties dès qu'il prétend les posséder.

3) Le vérificateur vérifie la partie qu'il a choisie. Si elle correspond à une information (non-déterministe) sur x, dès lors il est convaincu à **50%** que le prouveur connaît x.

Si on répète cette procédure N fois, le vérificateur sera convaincu de la bonne foi du prouveur avec une probabilité précise de $1-(1/2)^N$. Le vérificateur choisit N à sa convenance, c'est à dire suffisamment grand.

IV.2.3 Rigueur dans la gestion interactive.

En fait, l'exigence d'une information non-déterministe est faible, et on peut constater que n'importe qui peut construire UNE des deux parties de la partition. Mais seul celui qui peut construire les deux parties, par définition, connaît x ! Il faut garantir au vérificateur que la partition a été réellement faite et que le prouveur n'a pas la possibilité de la modifier en fonction du choix qui sera effectué par le vérificateur, et cela sans que le vérificateur ne puisse voir le contenu des parties.

On utilise une variante du protocole interactif du **double cadenas** [fig IV.1].

1) Le prouveur constitue les deux parties de l'information x, soient A et B.

2) Il place A dans un coffre à double cadenas et il boucle A avec la clé Ap.

3) Il place B dans un second coffre à double cadenas et attaché au premier ; il boucle B avec la clé Bp.

4) Il envoie les deux coffres scellés au vérificateur.

5) Le vérificateur boucle les deux coffres avec ses clés : soient respectivement Av et Bv de sorte qu'à ce stade ni le prouveur ni le vérificateur ne peuvent ouvrir les coffres sans l'accord de l'autre.

6) Le vérificateur choisit au hasard l'ouverture de A ou B et signale son choix au prouveur en renvoyant les deux coffres.

7) Le prouveur enlève son cadenas du coffre choisi et renvoie les deux coffres. Il ne peut pas tricher car il ne peut retirer ni Av ni Bv ; et ces clés permettent au vérificateur d'identifier si la partie choisie est réellement celle qui a été ouverte par le prouveur.

8) Le vérificateur ouvre la partie choisie. Il ne peut pas retrouver x car il ne peut ouvrir la partie qu'il n'a pas choisie. Il vérifie si l'information non-déterministe sur x est valide.

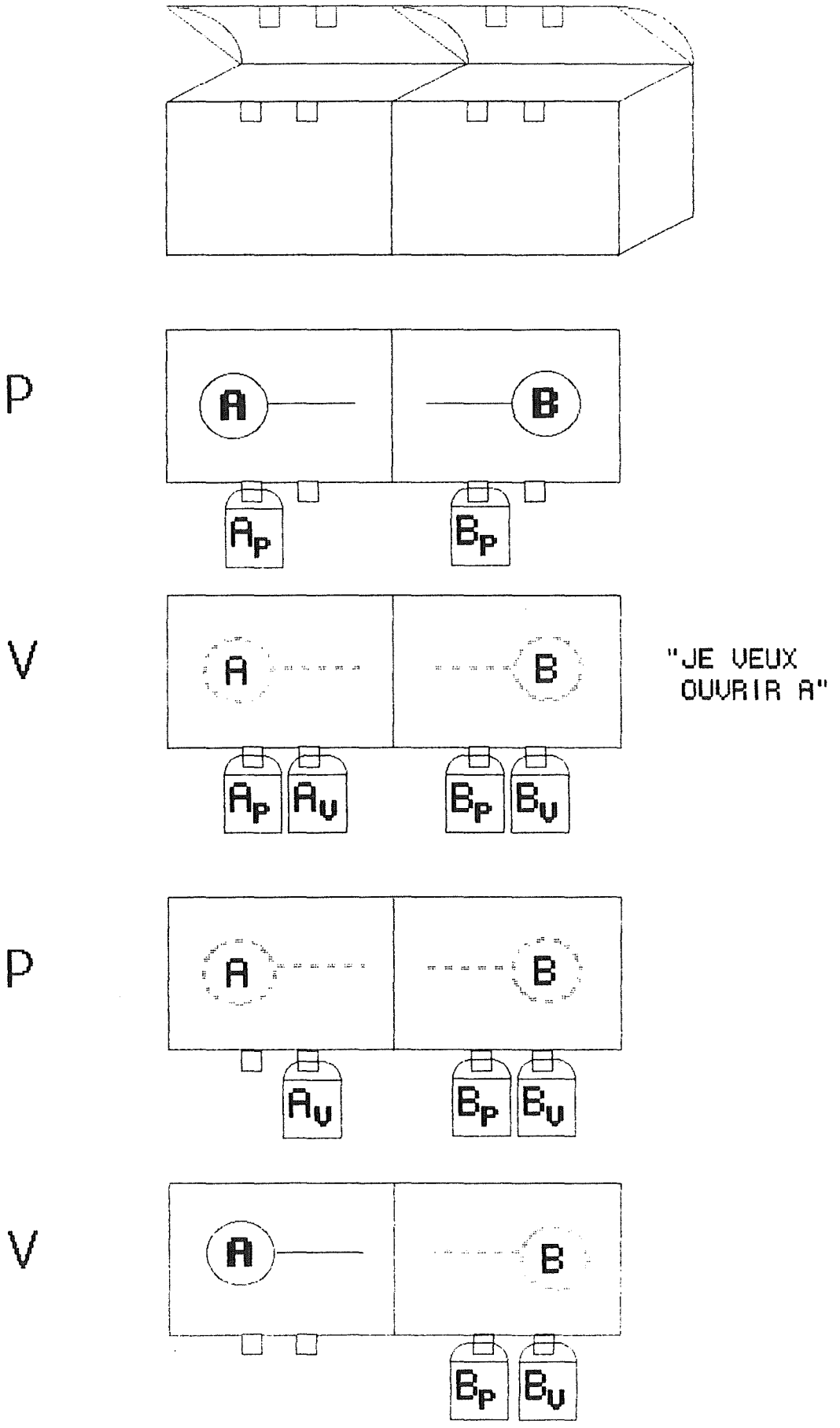


FIG IV.1 PROTOCOLE DU DOUBLE CADENAS.

IV.2.4 BLOBS et "Bit Commitment Schemes" (BCS).

Le protocole du double cadenas (généralement avec un seul coffre) détermine ce qu'on appelle un BLOB. La notion de BLOB [Brassard & Crépeau, 86] correspond intuitivement à un gage ou mieux une enveloppe scellée. Le prouveur donne au vérificateur une enveloppe munie d'un cachet de cire et contenant une information de telle sorte que ce dernier ne puisse l'ouvrir qu'en présence du prouveur. Au moment de l'ouverture, si on constate que le cachet est défait, c'est qu'il y a eu tricherie de la part d'un des deux partenaires.

Ici surgit une difficulté informatique : comment construire l'enveloppe et le cachet ? Constatons d'abord qu'on peut réduire le problème au cas où l'information transmise est un seul bit, ce qui correspond à un "Bit Commitment Scheme". En effet, toute information sera décomposée en bits; il suffit donc d'un nombre d'enveloppes BCS correspondant à la longueur binaire de l'information.

Les BCS sont réalisés pour l'instant par des fonctions trappes. Des exemples sont exposés dans [Brassard & Crépeau, 86]. L'avantage de leur utilisation est de permettre la construction de protocoles interactifs se basant sur leur existence théorique. On ramène la possibilité de réalisation de zero-knowledge pour tout problème NP-Complet à celle de l'existence de BCS [Brassard, Chaum & Crépeau, 87]. Il faut cependant faire attention à ne pas tourner en rond car un BCS n'est qu'un zero-knowledge élémentaire déguisé et de plus généralement basé sur des problèmes NP dont on ne sait pas si ils sont NP-Complets ! C'est le cas, par exemple, de l'isomorphisme de graphes utilisé dans le zero-knowledge par circuit hamiltonien [cfr IV.3.2]. Une question ouverte est de savoir si la cryptographie a intérêt ou non à utiliser des problèmes classés dans NP\(P U NP-Complet) [Goldwasser, Micali & Rackoff, 86].

IV.3 EXEMPLES DE PROTOCOLES "ZERO-KNOWLEDGE".

IV.3.1 Isomorphisme de graphes.

On veut prouver que deux graphes sont isomorphes, et ceci sans montrer l'isomorphisme [Goldreich, Micali & Widgerson, 86].

Soient G1, G2 tels que G1 ISO G2.

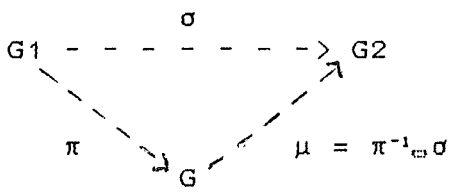
Démontrer que deux graphes sont isomorphes est un problème NP. Ce problème est très complexe si le nombre de noeuds des graphes est élevé.

Le prouveur peut prouver l'isomorphisme entre G1 et G2 si et seulement s'il connaît la permutation σ sur les noeuds telle que G2 = σ(G1). Mais en fait, il ne veut pas dévoiler σ. Dès lors, il opère comme suit :

- Il construit une permutation aléatoire π sur les noeuds de G1.

- Il peut construire G = π(G1).

- Il peut construire $\mu = \pi^{-1} \circ \sigma$.



- Il montre G au vérificateur (et G sert de BLOB).

- Le vérificateur choisit une des démarches suivantes:

  1) : le prouveur montre que G = π(G1).

  2) : le Prouveur montre que G2 = μ(G).

- Le prouveur répète la procédure en construisant une nouvelle permutation π de G1 jusqu'à conviction du vérificateur.

Le protocole est basé sur le partitionnement de l'information σ en $(\pi, \pi^{-1} \circ \sigma)$ dont les constituants sont des informations non-déterministes dans le modèle P (P-aléatoires).

IV.3.2 Graphe hamiltonien.

IV.3.2.1 Idée de base et solution intuitive (La Recherche, Fév. 87).

On veut prouver qu'un graphe est hamiltonien c'est à dire qu'il possède un chemin qui passe par tous les sommets une et une seule fois. Montrer qu'un graphe est hamiltonien est un problème qui est NP-Complet dérivé de HC [cfr IV.1.2].

Le principe est semblable à celui de l'exemple IV.3.1 :

Soit G* le graphe dont il faut montrer qu'il est hamiltonien. Le prouveur construit G isomorphe à G* par une permutation aléatoire des noeuds, soit σ. Si on connaît un chemin hamiltonien h de G*, alors forcément h'= σ(h) est un chemin hamiltonien de G.

Le protocole est simple:

- Le prouveur montre G.
- Le vérificateur choisit une des démarches suivantes :
  1) le prouveur montre que G est isomorphe à G*;
     le prouveur montre donc σ.
  2) le prouveur montre que G est hamiltonien;
     le prouveur montre donc h'.
- Le prouveur répète la procédure en construisant une nouvelle permutation σ de G* jusqu'à conviction du vérificateur.

Le partitionnement, contrairement à l'exemple précédent, est tel que chaque partie est de nature différente : respectivement une permutation et un graphe hamiltonien.

NOTE : ce protocole est faux.

En effet, il semble contenir plus qu'un seul bit d'information parce qu'il y a deux niveaux de complexité : celui du graphe hamiltonien qui est NP-Complet et celui de l'isomorphisme de graphe dont on ne sait pas si il est NP-Complet ! Il est possible que montrer un chemin hamiltonien sur un graphe isomorphe apporte de l'information déterministe sur le graphe original [Goldwasser, Micali & Rackoff, 86].

IV.3.2.2 Protocole de Blum [Blum, 86].

Le protocole décrit précédemment n'est pas zero-knowledge. Pour y remédier, Blum propose de ne pas révéler le graphe isomorphe lorsqu'on montre le chemin hamiltonien. Pour ce faire il utilise des "boîtes" (boxes) qui sont en fait des BLOBS.

Soient les n noeuds du graphe G, numéroté N1,..,Nn.

Le prouveur choisit un circuit hamiltonien dans G.

Par exemple : n=4 ; G=[(N1,N2),(N2,N3),(N3,N4),(N4,N1)]

Par simplification G est déjà un circuit hamiltonien !

Le protocole est :

- Le prouveur choisit aléatoirement une des n! permutations des n noeuds N1,..,Nn et place respectivement les noeuds permutés dans n boîtes B1,..,Bn. Par exemple : B1=[N2], B2=[N3], B3=[N1], B4=[N4].

- Pour chaque paire de boîtes (Bi,Bj), le prouveur place dans une nouvelle boîte Bij : 1 si le noeud placé dans la boîte Bi est relié par une arête du graphe G au noeud placé dans la boîte Bj, et 0 sinon. Dans l'exemple : B12=[1], B13=[0], B14=[1], B23=[1], B24=[0], B34=[1]. Il y a n*(n-1)/2 boîtes Bij.

- Il y a donc au total n + n*(n-1)/2 = (n+n²)/2 boîtes.

- le vérificateur choisit une des démarches :

  1) le prouveur montre au vérificateur que les boîtes contiennent une représentation correcte de G (graphe isomorphe en fait). Le prouveur ouvre **toutes** les boîtes.

  2) le prouveur montre que la représentation (du graphe G ou non !) contient un circuit hamiltonien. Le prouveur ouvre exactement n boites Bij,Bjk,..Bmi. Le vérificateur vérifie que ces boîtes contiennent toutes des 1 et que leur numérotation forme un cycle. Donc, dans l'exemple, le prouveur montre B12, B23, B34, B41.

- Le prouveur répète la procédure en construisant une nouvelle assignation de boîtes aux noeuds jusqu'à conviction du vérificateur.

IV.3.3 Résidus quadratiques (protocole de Fiat-Shamir) [Fiat & Shamir, CRYPTO' 86]).

Soient p et q premiers et grands tels que N = p * q. On dit que y est un **résidu quadratique** si et seulement si :

$$y = w^2 \bmod N \quad \text{où } w \text{ est premier avec } N \ (w < N).$$

Retrouver w à partir de la seule connaissance de y et N est un problème NP-Complet [Garey & Johnson, 79] qui peut se réduire au problème de la factorisation de N et dont la complexité est exponentielle.

On veut prouver que y est un résidu quadratique sans montrer w.

Protocole:

- Le prouveur choisit u premier avec N (u < N) de façon aléatoire. Il calcule z = u^2 mod N. Puis il rend public le résidu quadratique z ainsi calculé.

- Le vérificateur choisit une des démarches suivantes :

  1) le prouveur montre u de telle sorte que le vérificateur puisse vérifier que z est réellement un résidu quadratique.

  2) le prouveur montre v = (u*w) mod N de telle sorte que le vérificateur puisse vérifier que :

$$v^2 = (u*w)^2 = u^2*w^2 = z*y \pmod N.$$

  Le protocole fonctionne par la propriété des résidus quadratiques :

  Si a = b * c mod N où a et b sont des résidus quadratiques , alors c est aussi un résidu quadratique par composition des symboles de Jacobi [Riesel, 85: Appendix A], [Hardy & Wright, 79, p.73-78], [Goldwasser, Micali & Rackoff, 86 p.11], [BYTE, Oct. 87].

  Il suffit de considérer a = v^2 mod N et b = z car z est résidu quadratique dans l'hypothèse du second cas, dès lors c = y est également un résidu quadratique. Et pour construire v, il fallait connaitre w.

- Le prouveur répète la procédure en construisant un nouveau résidu quadratique z à partir d'un u aléatoire, jusqu'à conviction du vérificateur.

Le protocole de Fiat-Shamir est identique à cet exemple, mis à part le fait qu'il est effectué "en série", c'est à dire qu'on génère plusieurs u en même temps et qu'on effectue la vérification sur le produit. Il faut cependant être prudent quant à la procédure interactive de démonstration (notion d'interactivité en parallèle ?).

Pour plus de renseignements, on se réfèrera au chapitre consacré aux implémentations. Ce protocole est implémenté à titre démonstratif sur de grands nombres [cfr VII.2].

IV.4 "BIT COMMITMENT SCHEME" (BCS) OPERATIONNEL.

IV.4.1 Généralités.

Tout protocole zero-knowledge possède quelque part un BCS, et dans les exemples simples il est souvent présent de façon inhérente si bien qu'il est difficile, voire impossible, de l'extraire. Néanmoins, extraire le BCS possède l'avantage de décomposer le problème en couches. On peut ainsi se confectionner un jeu de BCS qui sont indépendants des zero-knowledge supérieurs qui les utilisent. De plus, il existe des classes de BCS sur lesquelles on peut définir des opérations précises. Il est très intéressant d'effectuer des calculs non fraudés **(dits honnêtes)** sur des bits dont on ne connaît pas la valeur.

En fait, un BCS est un zero-knowledge élémentaire dans lequel, dans une première partie, **les rôles** de prouveur et de vérificateur **sont inversés** pour éviter que le vérificateur ne triche aussi !

En fait, les BCS peuvent être inconditionnellement sûrs pour le prouveur ou pour le vérificateur. Tout dépend de l'exigence et de la réalisation de l'application.

IV.4.2 BCS à partir des résidus quadratiques (ou "Quadratic Residuosity Scheme" : QRS).

Soit le vérificateur qui calcule N = p * q où p et q sont de grands nombres premiers, comme précédemment. On procède comme suit :

- Soit B , le bit que veut cacher le prouveur.

- Le vérificateur choisit t de façon aléatoire (t inférieur et premier avec N) et montre au prouveur $s = t^2 \bmod N$.

- Le prouveur vérifie par une procédure interactive de dévoilement minimum (ou zero-knowledge) que s est effectivement un résidu quadratique [idem que IV.3.3].

- Le prouveur choisit y inférieur et premier avec N. Il calcule $x = y^2 * s^B \bmod N$. x est bien un résidu quadratique car s^B l'est (car B = 0 ou 1).

  Il est impossible pour quelqu'un d'autre que le prouveur de retrouver si x vient d'un produit ou non.

  x est le BLOB impossible à ouvrir sans l'accord du prouveur, mais ce dernier ne peut plus changer sa valeur !

- Le prouveur ouvre le BLOB sur demande du vérificateur en donnant y.

- Le vérificateur vérifie que :

$B = 1$ SSI $x = y^2 * s \bmod N$
$B = 0$ SSI $x = y^2 \bmod N$
Le prouveur triche sinon.

Ce protocole est inconditionnellement sûr pour le prouveur, car le vérificateur ne peut pas tricher. Par contre, pendant le temps de la transaction (et là seulement), il est possible que, par chance, le prouveur trouve le résidu quadratique de s. Le BCS qui suit possède la même propriété [cfr IV.4.3.3].

IV.4.3 BCS à partir de logarithmes discrets (ou "Discrete Logarithm Scheme" : DLS).

IV.4.3.1 Définitions.

- α est un **générateur** de Z^*p , le groupe multiplicatif des entiers modulo p , si et seulement si pour tout $k \in Z^*p$ il existe un et un seul $S \in Z^*p$ tel que $k = \alpha^S \pmod p$.

- Le problème du **logarithme discret**, comme son nom l'indique, consiste à inverser l'exponentielle dans un espace discret. Ce problème est supposé dans NP parce que intraitable. Il faut donc retrouver S à partir de k, ce qui par propriété des congruences peut se ramener à retrouver l'unique $0 < E < p-1$, tel que $S = \alpha^E \pmod p$.

IV.4.3.2 Blob.

Le blob contenant le bit B peut être construit d'une manière similaire à l'exemple précédent:

- Le vérificateur choisit $E \in Z^*p$ de façon aléatoire de telle sorte que $S = \alpha^E \pmod p$ et $S \in Z^*p \setminus \{1\}$. Il donne S au prouveur.

- Le prouveur choisit $Y < p-1$ de façon aléatoire et calcule le blob $x = S^B * \alpha^Y \bmod p$. Ensuite, il donne x au vérificateur.

  Il s'agit réellement d'un blob : en effet, le prouveur ne peut trouver Y1 différent de Y2 de telle sorte que $\alpha^{Y1} = S * \alpha^{Y2} \pmod p$ car sinon $\alpha^{Y1} = \alpha^{E+Y2}$ ou encore $Y1-Y2 = E \pmod p$; or le prouveur ne peut retrouver E par propriété d' "intraitabilité" des logarithmes discrets.

- Le prouveur ouvre le blob sur demande du vérificateur en donnant Y.

- Le vérificateur vérifie que :

$B = 1$ SSI $x = S * \alpha^Y \bmod p$
$B = 0$ SSI $x = \alpha^Y \bmod p$
Le prouveur triche sinon.

IV.4.3.3 Remarques.

Cette implémentation ne sera pas correcte si on ne permet pas au prouveur de vérifier que S est effectivement calculé à partir d'une exponentielle. En effet, le vérificateur pourrait tricher en calculant par exemple $S = 2*\alpha^{E} \pmod p$, si le protocole est effectué en parallèle (une seule interaction) et qu'il possède une table des logarithmes discrets de 2.

On ne peut donc parler de parallélisme qu'après que le vérificateur ait prouvé au prouveur qu'il connait le logarithme discret de S. Sinon les blobs ne sont inconditionnellement sûrs que vis-à-vis du prouveur. (On peut en construire d'autres en utilisant les propriétés des symboles de Jacobi qui seront inconditionnellement sûrs pour le vérificateur [Brassard & Crépeau, 86], [Chaum, Evertse & al., CRYPTO'86]).

IV.4.4 BCS à partir d'isomorphisme de graphes.

Par le même principe, le vérificateur prend le rôle de prouveur pendant un instant afin de travailler sur des propriétés qui ont pour but d'empêcher les deux partis de tricher :

- Le vérificateur montre que deux graphes E et F sont isomorphes par une procédure de dévoilement minimum de l'isomorphisme σ.

- Le prouveur construit le blob comme étant un graphe, et ce en fonction du bit B qu'il veut sceller :

  Si B = 0 : graphe isomorphe à E (construit μ).
  Si B = 1 : " " à F (construit τ).

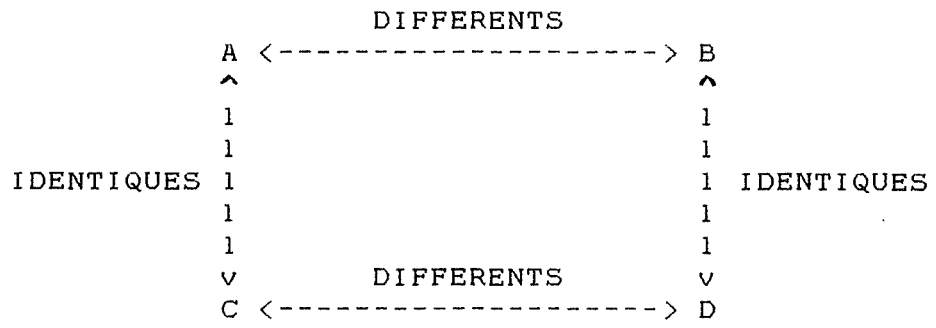
  Le prouveur ne peut pas tricher car sinon il pourrait à la fois construire les isomorphismes μ et τ et donc il connaîtrait σ, ce qui n'est pas possible parce que ce problème est intraitable.

- Pour ouvrir le blob, le prouveur montre μ ou τ.

IV.4.5 Comparaison de blobs.

Le problème qui se pose ici consiste à vérifier que deux blobs sont identiques (ou différents) et ce **sans les ouvrir** !

IV.4.5.1 Théorème d'équivalence.



Le problème d'égalité de blobs est équivalent à celui d'inégalité.

A) Egalité => inégalité.

Si on veut montrer au vérificateur que le contenu du blob A est différent de celui de B :

- Le prouveur construit 2 blobs C et D dont les contenus sont différents et les montre au vérificateur sans les ouvrir.

- Le vérificateur choisit 1) ou 2) :

  1) Le prouveur montre que le contenu de A est égal au contenu de C (ou D) et que le contenu de B est égal au contenu de D (ou respectivement C). Ceci est possible, de par l'hypothèse qu'il existe un protocole permettant la vérification d'égalité de blobs sans les ouvrir.

  2) Le prouveur ouvre les blobs C et D afin de vérifier que leurs contenus sont réellements différents.

- On répète la procédure jusqu'à conviction du vérificateur, en montrant deux nouvelles paires (C,D) à chaque tour.

B) Inégalité => égalité.

Si on veut montrer au vérificateur que le contenu du blob A est égal à celui de C :

- Le prouveur construit 2 blobs B et D dont les contenus sont identiques et les montre au vérificateur sans les ouvrir.

- Le vérificateur choisit 1) ou 2) :

1) Le prouveur montre que le contenu de A est différent du contenu de B (ou D) et que le contenu de C est différent du contenu de D (ou respectivement B). Ceci est possible, de par l'hypothèse qu'il existe un protocole permettant la vérification d'inégalité de blobs sans les ouvrir.

2) Le prouveur ouvre les blobs B et D afin de vérifier que leurs contenus sont réellements identiques.

- On répète la procédure jusqu'à conviction du vérificateur, en montrant deux nouvelles paires (B,D) à chaque tour.

IV.4.5.2 Egalité de blobs par résidus quadratiques.

Si $x = y^2 * S^B \pmod N$ et $x' = y'^2 * S^B \pmod N$ représentent le même bit B ; dès lors, si le prouveur calcule $z = y*y'*S^B$ et le montre au vérificateur, ce dernier sera convaincu que x et x' représentent le même bit en vérifiant que $x*x' = z^2 \pmod N$.

IV.4.6 Opérations sur des blobs.

Il n'est pas possible d'effectuer des opérations sur tous les blobs. Par exemple, on ne voit pas comment définir des calculs à partir de l'isomorphisme de graphes. Ici on se base sur les blobs à partir de résidus quadratiques.

Soient B1 et B2, les deux bits secrets de A, ainsi que z1 et z2 leurs blobs respectifs du type résidu quadratique passés à B. B peut calculer une fonction sur les Bi à travers les blobs, et ce sans qu'il ne connaisse les bits qu'ils cachent !

IV.4.6.1 Négation de blobs.

On cherche le blob qui est la négation du blob z1. Il suffit de construire le blob $z = z1*y \bmod N$, car par construction, on a : $z = x^{2}*y^{(B1+1) \bmod 2} \bmod N$.

IV.4.6.2 "OU EXCLUSIF" de blobs.

Le "OU EXCUSIF" ne correspond pas à la "blinding property" (BP) [Chaum, Damgård & van de Graaf, CRYPTO'87] parce que pour cette dernière, l'opération s'effectue par B entre un blob créé par A et un bit connu de B seul et non pas entre deux blobs crées par A.

B calcule le blob $z = z1*z2 \bmod N$ qui cache en fait le ou exclusif du bit B1 du blob z1 et du bit B2 du blob z2 si $z1 = x1^{2}*y^{B1} \bmod N$ et $z2 = x2^{2}*y^{B2} \bmod N$:

On cherche $z = x^{2}*y^{(B1+B2) \bmod 2} \bmod N$. Le tout est de trouver x. Or, si on pose que le résidu quadratique vaut

$x = x1*x2*y^{(B1+B2) \operatorname{DIV} 2} \bmod N$, on a $z = x^{2}*y^{(B1+B2) \bmod 2} =$

$(x1*x2)^{2}*y^{(2*((B1+B2) \operatorname{DIV} 2) + (B1+B2) \bmod 2)} =$

$(x1*x2)^{2}*y^{(B1+B2)} = z1*z2 \pmod N$.

IV.4.6.3 Autres opérations.

Dans l'état actuel des connaissances, il n'a pas encore été établi si le ET de blobs ou une autre opération est possible. Autrement dit, entre blobs, les seules opérations connues sont la négation et le "OU EXCLUSIF" pour résidus quadratiques [Brassard & Crépeau, CRYPTO'86, pp.225-226].

IV.5 SECURITE MULTIPARTITE [CHAUM, DAMGARD & VAN DE GRAAF, CRYPTO'87].

Ce paragraphe a pour but de montrer une des applications possibles (plutôt future) des blobs. On s'intéresse à un protocole qui permet à un ensemble de partis (auquels des amis et adversaires participent) d'effectuer d'un commun accord des opérations (sur un réseau local, une machine partagée ou encore un système fault-tolerant, par exemple) de telle sorte que chaque parti puisse choisir des entrées secrètes et vérifier que chaque sortie correspondante est correcte; de plus, dans ce protocole, toutes les entrées secrètes sont protégées de façon **optimale**. Les entrées et les sorties correspondent de façon théorique aux bandes de lecture et d'écriture de machines de Turing. Il faut définir ce qu'on entend par optimalité.

Un participant est autorisé à cacher ses secrets de façon **inconditionnelle**, c'est à dire le protocole ne laisse passer aucune information au sens de Shannon (en fait, zero-knowledge). Ceci signifie qu'un participant peut effectuer des opérations de façon sûre avec un autre, même si ce dernier possède une puissance de calcul **illimitée**.

Selon Damgård [Damgård, 87, pp.48-66], ce protocole est le premier à permettre une sécurité **inconditionnelle** multipartite.

IV.5.1 Propriétés.

Le protocole est **ouvert** si n'importe qui peut, par la suite, interroger de façon interactive un participant afin de voir qu'aucune tricherie n'a été effectuée.

Le protocole est à apport minimal de connaissance car même si n-1 des n participants collaborent dans un but de récolter un maximum d'informations sur le dernier, ils ne récolteront rien de plus que ce qu'ils auraient pu récolter séparément à son propos (donc au moyen de la sortie publique).

L'avantage essentiel de cette approche est son indépendance vis-à-vis de la réalisation, par exemple, de fonctions trappes, et ceci essentiellement parce que cette difficulté est reportée à un niveau inférieur par l'utilisation de blobs théoriques introduits par les travaux de Chaum, Brassard et Crépeau [cfr IV.4.3].

L'approche modulaire permet d'ailleurs, moyennant une augmentation de coût négligeable, de prévoir que les participants puissent prendre connaissance **simultanément** de la sortie. Ceci fait référence aux travaux récents concernant le "gradual release".

En effet, si la puissance de calcul est infinie, la moindre avance du point de vue connaissance d'un des participants peut amener à des déséquilibres pouvant mettre en péril la sécurité du protocole (non I-aléatoire [cfr IV.2.2]). Le "gradual release" permet de synchroniser la communication d'information; ceci est rendu difficile à cause de l'interactivité qui est en fait un dialogue. Dans un dialogue qui consiste en un échange d'informations, l'un des deux participants recevra l'information totale de l'autre avant que l'autre ne reçoive toute celle du premier; or, c'est justement ce qu'on veut éviter [Brickell, Chaum & al., CRYPTO'87].

Un avantage théorique de cette méthode est qu'elle permet de démontrer que tout langage dans IP (ensemble des problèmes qui peuvent être prouvés de façon interactive) peut être prouvé par un protocole de type zero-knowledge.

IV.5.2 Idée de la construction.

Par simplification, on va considérer deux partis seulement : soient A et B. De plus, on s'intéresse à une seule porte "ET" pour commencer. Chacun des deux partis ne sait pas ce que l'autre veut vérifier.

IV.5.2.1 Préliminaires.

Tout d'abord, chacun des partis choisit un "Bit Commitment" de telle sorte que le BCS de A possède la "Blinding Property" (BP) :

cette propriété ne permet pas d'effectuer des calculs entre blobs, mais bien le "OU EXCLUSIF" d'un bit caché par un blob BP avec un bit donné, et d'encapsuler le résultat dans un nouveau blob BP.

Dans le paragraphe précédent, on a vu comment comparer des bits d'un même principe BCS. Par un procédé identique, il est possible de comparer des BCS construits différemment.

La porte "ET" est représentée par sa table de vérité T.

On cherche des transformations appliquées à T par A et B consécutivement, de telle sorte que :

- les deux partis peuvent être sûrs (avec une très grande probabilité) que les transformations de l'autre sont effectuées correctement.

- chaque parti peut garder son entrée secrète.

- chaque parti peut être assuré de l'exactitude des opérations qu'il effectue.

IV.5.2.2 Protocole.

Transformation de la table T par A.

1.1 Application par A d'une permutation aléatoire des lignes de T.

1.2 Inversion de colonnes.

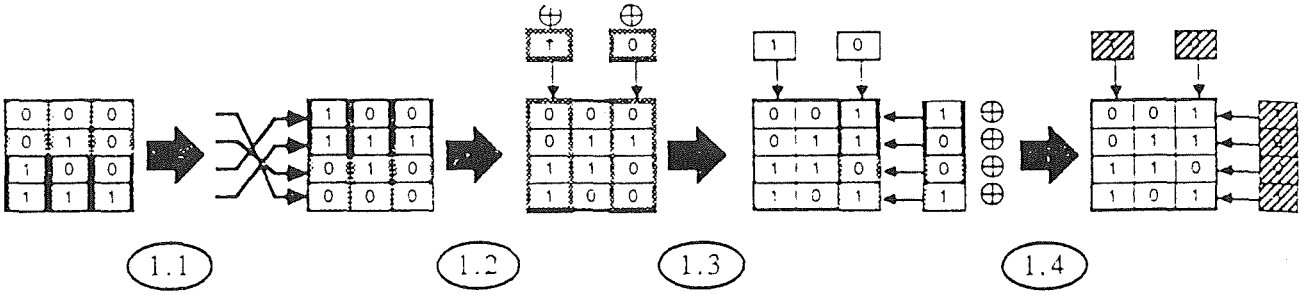
B choisit aléatoirement 2 bits b1 et b3 et effectue le "OU EXCLUSIF" de la K^{IEME} colonne avec bK.

1.3 Codage de la colonne de sortie.

A choisit aléatoirement 4 bits d1,...,d4 et effectue le "OU EXCLUSIF" de la dernière colonne correspondant à l'entrée L avec le bit dL.

1.4 Calcul par A des BCS correspondant à b1,b3,d1,...,d4. On représente les blobs de façon hachurée [fig IV.2].

La table résultat est notée T^.



Party *A*'s transformation of the AND-gate.
Note that all figures show the table *after*
the indicated operation has been performed.

FIG IV.2 TRANSFORMATION D'UNE PORTE "ET" PAR A.

B vérifie si A a construit correctement T^.

Pour ce faire, A crée une table T' construite de la même façon.

B choisit 1) ou 2).

Si 1)

A doit révéler comment il a construit T' de telle sorte que B vérifie si A l'a fait correctement.

Si 2)

A doit montrer une relation spéciale entre T' et T^.

Pour ce faire, consulter [Damgård 87, pp.58-59].

Il faut que T' correcte + relation => T^ correcte.

Mais la relation n'apporte rien quant à la détermination de T^.

On répète en construisant autant de T' que ne l'exigera B pour être convaincu.

Transformation de la table T^ par B.

B utilise la "Blinding Property" des blobs de A afin de ne laisser qu'un seul "commitment" par ligne. De ce fait, même A ne connaît plus les correspondances entre les lignes puisque l'ouverture du nouveau blob ainsi calculé ne lui apporte plus aucune information sur les lignes.

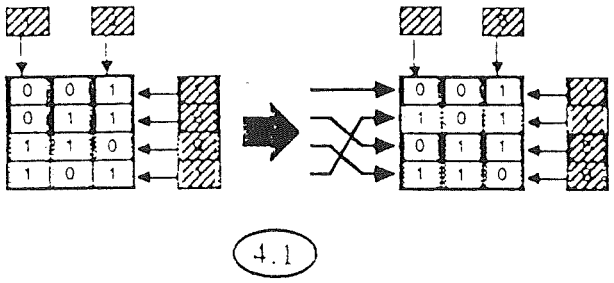
Le "Blinding" est nécessaire pour cacher la permutation de A (et remettre ainsi la table dans un état semblable à l'état initial).

Ensuite, B opère comme A :

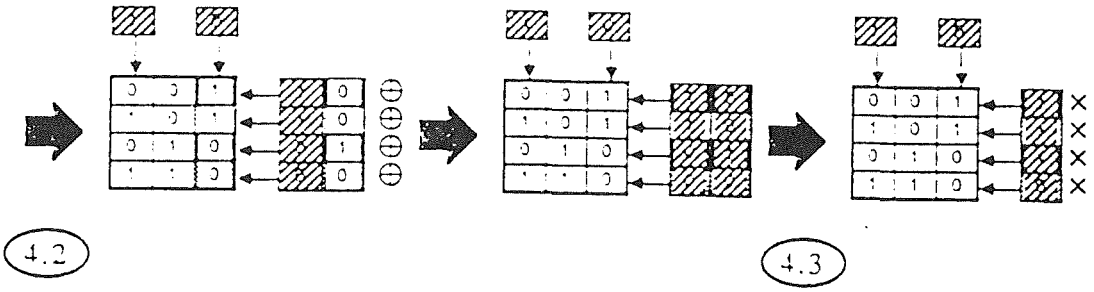
B inverse les lignes.

B doit également inverser sa colonne entrée et la colonne de sortie. De cette manière, B transforme T^ en T*.

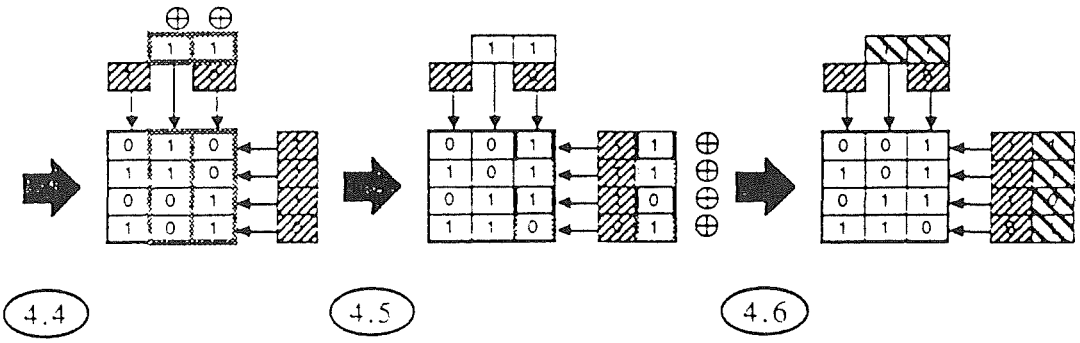
Enfin, B "commet" ses bits d'inversion [fig IV.3].



The row-permutation performed by *B*.



The blinding.



The final part of *B*'s transformation.

FIG IV.3 TRANSFORMATION PAR B.

A vérifie si B a construit correctement T'.

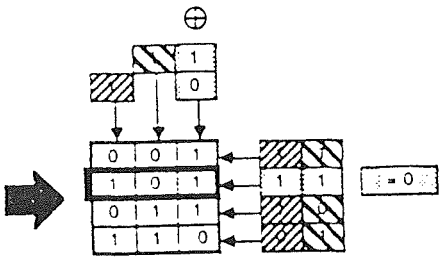
A vérifie que T^ -> T' est correct, de la même façon que B vérifie que T -> T^ est correct.

Calculs.

A et B ont donc construit T', version "double-blinded" de T. Il vont travailler sur cette table sûre.

Ensemble, ils choisissent la ligne de T' de façon correcte en annonçant leurs bits d'entrée de T'. Par construction, une seule sortie correspond.

A et B montrent les bits d'inversion de la colonne de sortie ainsi que ceux correspondant à la ligne choisie. On peut de cette manière calculer avec 100% de fiabilité le bit correspondant à la sortie, de telle sorte que les deux partis soient convaincus de la bonne foi de l'autre [Fig IV.4].



The computation.

FIG IV.4 CALCUL DANS LA TABLE.

Cas général.

Le protocole de base peut être étendu dans plusieurs directions.

- Plutôt qu'une porte "ET", n'importe quelle porte peut convenir.

- A la place de deux partis, on peut considérer N partis. Il suffit pour cela que les autres appliquent la BP sur la table permutée de la même façon que B.

- Plutôt qu'une seule porte, on peut construire n'importe quel circuit imbriquant un nombre aléatoire de portes. Les portes sont connectées de la manière suivante :

  soit la sortie de T1 connectée à une ou plusieurs entrées de T2 de telle sorte que les sorties et entrées correspondantes subissent le "OU EXCLUSIF" des mêmes bits, selon la convention dans les circuits booléens.

IV.6 TRANSFERT EQUIVOQUE ("OBLIVIOUS TRANSFER" (OT) [BLUM, 81]).

Le concept général de transfert équivoque a été introduit par Blum. Il fait partie, en quelque sorte, du mode de pensée induit par (et même antérieur à) l'idée de protocoles zero-knowledge. Mais l'objectif et la réalisation sont complètement différents. En effet, OT peut être vu comme un protocole de transfert d'une quantité non négligeable de connaissance avec probabilité 1/2. Il n'est donc évidemment pas un zero-knowledge.

Exemple [Goldwasser, Micali & Rackoff, 85 pp.302-303] :

Soient les deux partis A et B, ainsi qu'un entier n, produit de deux grands nombres premiers distincts et dont la factorisation n'est connue que par A.

A désire communiquer la factorisation de n à B de telle sorte que :

- B recevra la factorisation de n avec une probabilité 1/2. Dans le cas où il ne reçoit pas la factorisation, il n'a reçu aucune information lui permettant de la déduire; autrement dit il ne possède pas plus d'information qu'avant le protocole.

- A, le protocole terminé, ne sait absolument pas si B a reçu ou non la factorisation de n.

D'où A ne sait pas si l'information qu'il communique est redondante ou non.

Réalisation.

Pas 1:

- B choisit de façon aléatoire un entier x strictement compris entre 1 et n, et premier avec n.

- B calcule $y = x^2 \bmod n$ et envoie y à A.

Pas 2:

- A calcule une racine carrée (modulo n) de y, soit z, qu'il communique à B. Il peut le faire, puisqu'il connaît les facteurs de y.

Pas 3:

- B vérifie que $z^2 = y \bmod n$.

- Il est bien connu que tout nombre entier possède, dans un espace discret, 4 racines carrées généralement distinctes. Soient (x,-x,w,-w) les 4 racines de y. Par construction, la racine x est connue de B.

- Avec probabilité 50%, z sera w ou -w; dans ce cas, par des propriétés de la théorie des nombres, le PGCD(n,x+z) sera un facteur de n. D'où B sera à même de factoriser n.

Concernant la rigueur du protocole, consulter [Goldreich, Micali & Widgerson, 86 4.1].

DEUXIEME PARTIE
****

APPLICATIONS ET LIMITES.

CHAPITRE V

APPLICATIONS A LA CARTE A MICROPROCESSEUR.

V.1 EXIGENCES VIS-A-VIS DE LA CARTE.

V.1.1 "Zero-knowledge" et carte.

Dans les chapitres précédents, on a montré quels étaient les nouveaux protocoles théoriques ainsi que les tendances induites par ces nouveaux développements. On est en droit de se demander quelles sont les espérances en cryptographie et plus généralement en sécurité, d'un point de vue pratique. A ce propos, les idées théoriques précédentes ont toujours eu en vue des applications, notamment pour les systèmes de carte à mémoire. Chacune de ces idées répond à une attaque précise et il est bien évident qu'une application effective devra englober des protocoles précis à des niveaux différents en fonction du type de service requis et de la qualité de sécurité exigée.

Le but n'est pas ici de concevoir un nouveau type de carte à mémoire, mais de détacher les protections logicielles, d'un type nouveau, des protections physiques. Ceci sera surtout développé dans le cadre de l'identification.

Dans un premier temps, on analyse la première application qui vient à l'esprit.

Considérons le protocole du double cadenas [cfr IV.2.3]. On remarque d'abord qu'il possède les propriétés d'un blob. Dans tous les protocoles, on retrouve le dialogue entre le prouveur et le vérificateur. En fait, le microprocesseur de la carte fait généralement office de prouveur et le "lecteur" de la carte fait office de vérificateur ou mieux l'ordinateur à l'autre bout de la ligne. Mais, comme on l'a vu d'ailleurs pour des phases d'initialisation entre autres, les rôles peuvent être inversés.

Comment l'identification est-elle rendue possible?

Tout d'abord, il faut une phase d'initialisation auprès d'un centre, une autorité, en lesquels l'individu puisse avoir confiance. Chaque individu x se choisit une signature secrète, en accord avec le centre. Cette signature correspond au SID. En vue d'une sécurité inconditionnelle, il faudrait que le centre "oublie" cette transaction. Un peu comme les BCS de logarithmes discrets : le risque de tricherie n'est présent que lors de la transaction. L'idéal serait que l'individu calcule lui même son SID. Mais dans ce cas, si on ne peut plus se faire passer pour cet individu, ce dernier peut tricher sur son identité à lui.

Le centre a pour but d'établir ce qu'on peut appeler la **formule**. Ceci est en rapport avec le fait que tout problème NP-Complet peut se ramener à SAT. La formule consiste en un problème dont la solution est réputée difficile, c'est-à-dire Non déterministe Polynomiale au sens de la théorie de la complexité. La solution correspond à la signature secrète. Le couple <x, formule(ID(x))> est ajouté à la table publique. Une fois la phase d'initialisation terminée, l'identification peut commencer [fig V.1].

Classiquement, l'identification consistait à vérifier si une information se trouvait effectivement dans la carte ; on demandait en fait le contenu d'une adresse précise dans la carte - ce contenu étant propre à la carte. Le désavantage est qu'il faut demander à chaque fois un certificat différent, or la mémoire est limitée. De plus, et surtout, le certificat passe sur la ligne !

Seule la signature secrète permet de décomposer en deux parties (1) et (2) le problème, comme indiqué au chapitre précédent avec les trois propriétés :

A) (1) + (2) permet de retrouver SID.

B) (1) n'apporte rien sur SID.

C) (2) n'apporte rien sur SID.

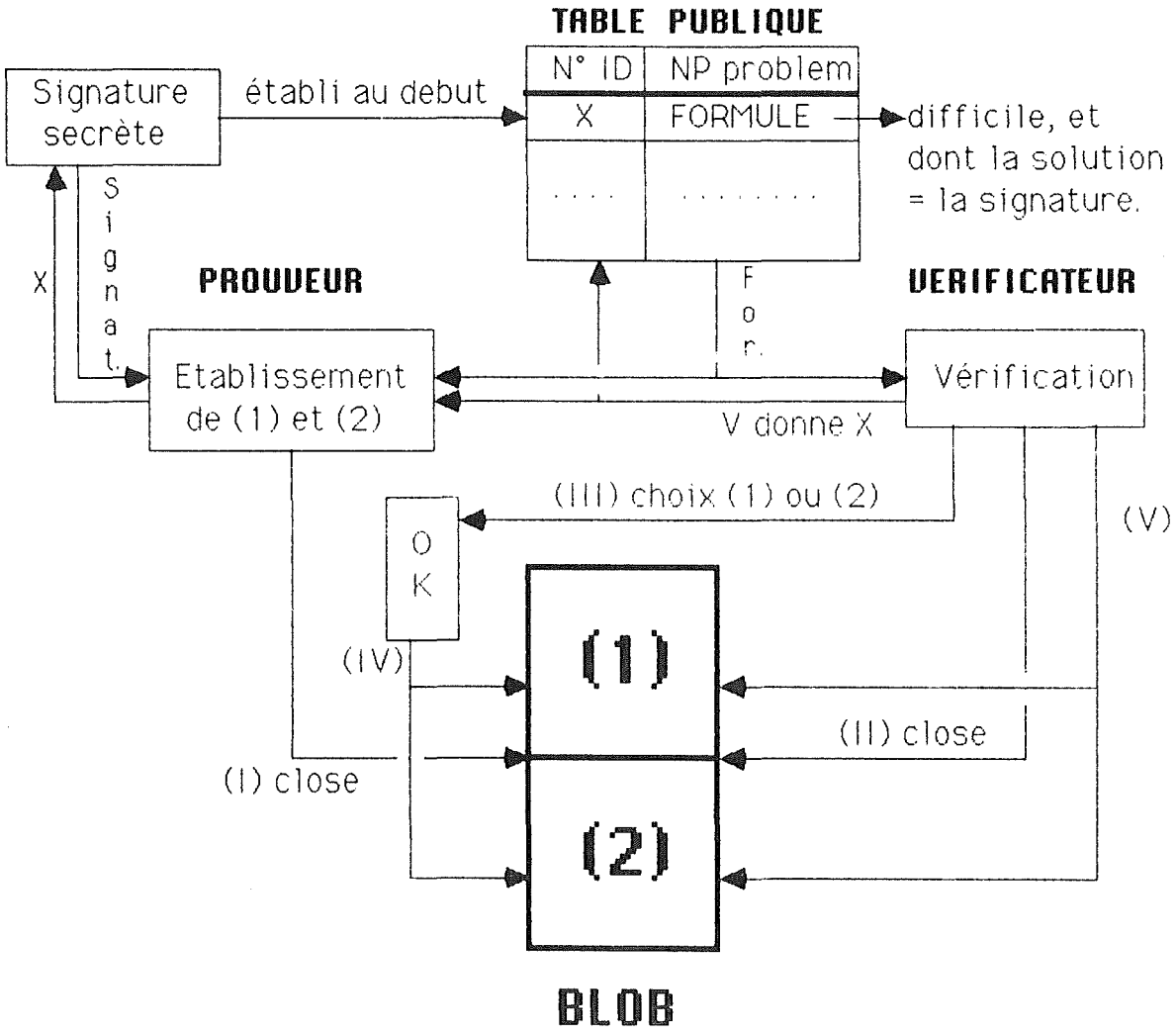


FIG V.1 APPLICATION DU PROTOCOLE DU DOUBLE CADENAS.

D'abord, V (vérificateur) donne x, ce qui permet à la fois à V et à P (prouveur) de prendre connaissance de la formule.

(*) Le prouveur divise la preuve de la satisfaisabilité de la formule en deux parties en établissant (1) et (2). Ensuite P place respectivement (1) et (2) dans chacune des deux parties du coffre-blob.

Par (I), P ferme le double coffre.

Par (II), V ferme le double coffre.

Ainsi, ni P ni V ne peuvent changer le contenu du coffre (ce qui est le principe du blob).

Par (III), V choisit (1) ou (2) de façon exclusive.

Par (IV), P ouvre (1) ou (2) selon le choix de V.

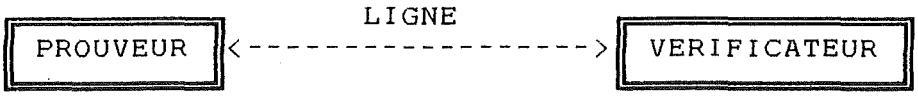
Et enfin, par (V), V vérifie que (1) ou (2) est correct par rapport au protocole utilisé.

On répète la procédure pour une nouvelle division du problème, à partir de **(*)**.

V.1.2 Sécurité logicielle.

V.1.2.1 Applications hiérarchisées.

On considère les schémas classiques d'authentification:



L'utilisateur doit prouver qu'il est réellement le détenteur du numéro d'identification ID. Pour ce faire, il détient son numéro d'identification secret SID. Ceci se fait par chiffrement :

chiffrement envoi déchiffrement
$\Xi(SID)$ ----------> $\Xi^{-1}(\Xi(SID)) = SID$

Ceci possède deux inconvénients majeurs :

- Ξ^{-1} est connue du vérificateur.
- SID est connu du vérificateur.

Généralement, le prouveur ne veut pas communiquer son SID car il est en concurrence avec tout vérificateur. Le prouveur veut éviter que le vérificateur puisse se faire passer pour lui. Tout réside dans le fait que Ξ^{-1} ne permet pas au vérificateur de retrouver Ξ.

Imaginons un système de télébanking. Comment assurer le client ou au moins la banque du client (prouveur), qu'aucun autre individu ni aucune autre organisation que cette banque à laquelle le client fait confiance, ne puisse avoir accès à son mot de passe ? Ceci est particulièrement utile dans des schémas logiciels indépendants de la carte même et donc du constructeur de cette dernière. En effet, la banque, qui généralement achète les cartes d'une autre société, désire une sécurité indépendante des propriétés même physiques de la carte.

Si on regarde la figure [fig V.2], les schémas logiciels sont représentés par les cadres X. En quelque sorte, chaque banque personnalise la carte. Supposons qu'un client de la banque A désire encaisser un chèque électronique auprès de la banque B au moyen d'une carte bancaire personnalisée par son SID, mais qu'il ne veuille pas dévoiler ce SID pour des raisons confidentielles. En fait, c'est surtout la banque A qui ne veut pas que la banque B, ou n'importe qui, le connaisse !

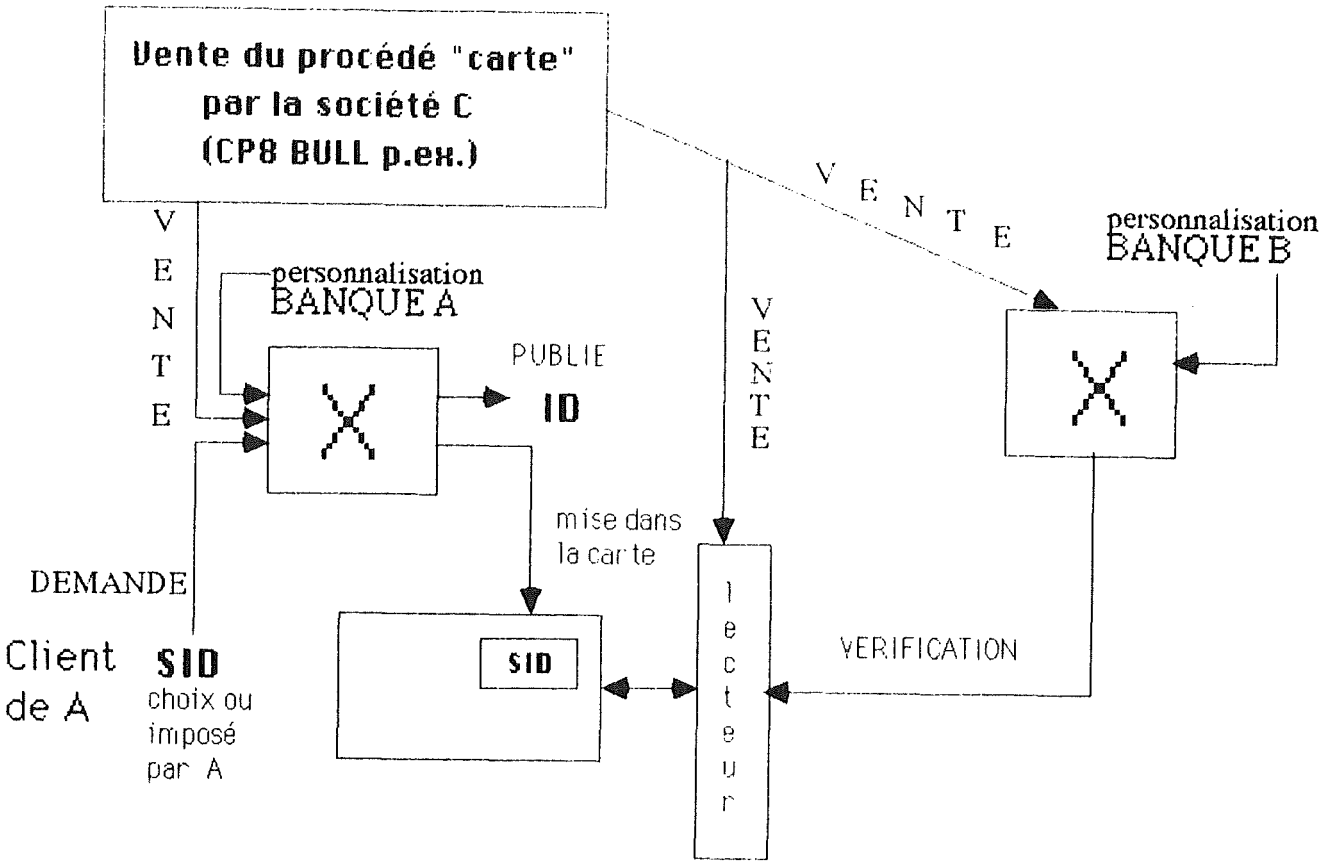


FIG V.2 HIERARCHIE DE SECURITE DANS UNE APPLICATION BANCAIRE.

Généralement, le SID du client sera d'ailleurs connu et même fourni par la banque A; dans ce cas, le client ne le connaîtra probablement même pas : il connaîtra seulement le PIN (cfr VI.2.2.4). Mais le protocole fonctionne encore si on envisage que le client de la banque A n'a même pas confiance en sa propre banque et qu'il garde ainsi son SID pour lui tout seul.

Les banques personnalisent les cartes par des classes possibles de logiciels de codage de telle sorte que ni les autres banques ni aucun individu ne puisse connaître SID.

Ceci est parallèle à l'idée de carte mère [De Pra, 87 p.59], mais ici chaque banque en a le contrôle pour ses clients propres. On peut même envisager qu'un client d'une banque soit une organisation qui ait son propre système X. Dans ce cas, il faut un accord "d'initialisation" entre cette organisation et sa banque.

Tout repose sur le fait que seul A connaît le lien entre ID et SID. Ceci nous reporte évidemment à l'idée de fonctions à sens unique, et au zero-knowledge.

V.1.2.2 Mécanismes d'accréditations en environnement multiparti.

On considère les mécanismes d'accréditations, qui peuvent être considérés comme un système de protocoles permettant de transmettre des informations personnelles (et donc confidentielles) entre organisations. On parle d'accréditations, par exemple pour des permis de conduire, cartes d'identité, cartes de crédit (au sens bancaire), etc... Ce sont en fait des informations sous forme de messages émanant de l'organisation, qui permettent d'affirmer ou d'infirmer l'accréditation. Il n'y a affirmation que lorsque l'individu est identifié.

Damgård propose deux exigences concernant les accréditations [Damgård, 87 pp.70-72]:

- aucun individu ne peut montrer une accréditation à personne excepté si elle lui est demandée correctement et par une organisation autorisée.

- les accréditations dispensées aux différentes organisations ne permettent pas à ces dernières, même si elles coopèrent, d'acquérir la moindre information conduisant à la reconstruction de l'image complète de n'importe quel individu.

Alors que les individus ont en général une puissance de calcul limitée, les organisations disposent de ressources importantes et on ne peut pas statuer sur leur puissance de calcul. Dans une certaine mesure, si elle est connue, elle peut servir d'**autorité**.

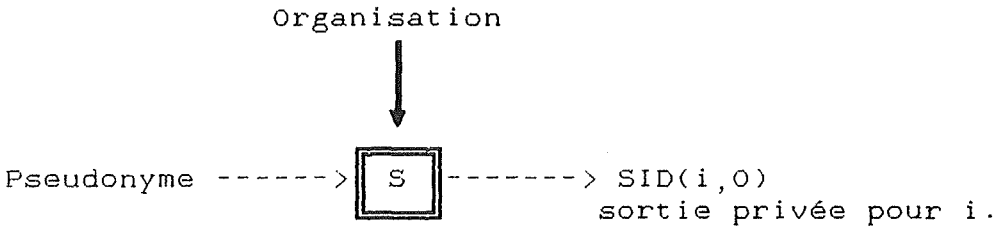
Chaum et Evertse [Chaum & Evertse, CRYPTO'86] ont présenté un mécanisme d'accréditations basé sur la sécurité du RSA dont la construction se révèle aisée au prix d'une autorité en laquelle il faut ramener toute la confiance des organisations et non pas des individus. Cette autorité a pour but de calculer toutes les accréditations.

De récents développements ont permis, de façon théorique pour l'instant, de réduire la confiance en l'autorité à une phase d'initialisation qui a pour but d'identifier les individus dans le système.

Ce dernier point est un avantage considérable par rapport à ce qui est fait actuellement car, pour les cartes, toute la confiance est à accorder au constructeur. Il serait bien plus intéressant que les organisations (généralement des banques), puissent introduire leur propre sécurité étanche vis-à-vis même du constructeur. Il est évident qu'une sécurité de ce type est logicielle, sinon la banque fait office de constructeur partiel et dans ce cas, elle doit posséder un matériel fourni par le constructeur dans lequel elle doit de toute façon ramener sa confiance.

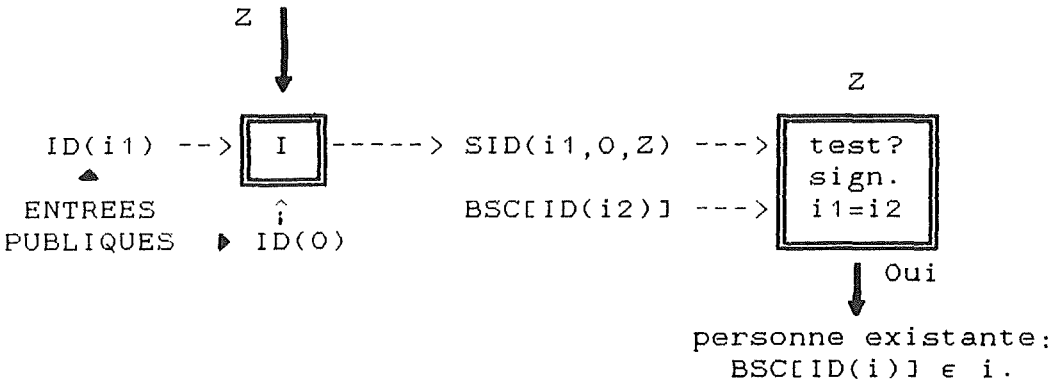
Soit ID(i), une information identifiant l'individu i. On fixe au départ les règles qui définissent les types d'accréditations en fonction des schémas de signature sur ID(i). Chaque organisation autorisée à vérifier ce type d'accréditations est supposée connaître la clé secrète correspondant au schéma de signature.

La solution proposée est de donner aux organisations non pas ID(i) mais le "commitment" des bits de ID(i) (càd les BCS). On le note BCS[ID(i)], on l'appelle **pseudonyme**. De ce fait, l'organisation signe le pseudonyme par sa clé secrète et donne en sortie un résultat privé pour i.



Ce procédé n'est malheureusement pas suffisant : en effet, n'importe qui peut se faire passer pour i; autrement dit, i peut générer plusieurs pseudonymes ! Pour éviter cela, il faut un centre Z (autorité en laquelle les organisations ont confiance) qui certifie que le pseudonyme identifie bien i.

L'avantage de cette nouvelle méthode réside dans le fait que cela peut être effectué une fois pour toutes. Il faut donc "personnaliser" i. Pour plus de détails, on consultera [Damgård, 87 p72].



V.2 LE PROBLEME DU CANAL SUBLIMINAL.

Lorsqu'on commence à penser aux applications de protocoles zero-knowledge, surgissent de nombreux problèmes "parasites" qui, jusqu'à récemment, empêchaient de construire des applications efficaces. Autrement dit, ce n'est pas la "panacée".

Hormis les problèmes de calcul ou d'interactivité (développés au chapitre VII), subsiste, entre autres mais essentiel parce qu'il remet en cause le principe même de zero-knowledge, le problème du canal subliminal (subliminal channel [Bengio, Brassard & al., 87]). Il s'agit en fait de variantes du problème des prisonniers de Simmons [Simmons, CRYPTO'85].

Deux prisonnier désirent communiquer de façon privée, mais toute leur correspondance doit passer par un gardien qui peut la lire. En fait, rien n'empêche de communiquer de manière détournée; et ceci est parfaitement réalisable dans le cas où l'information passée n'a rien à voir avec celle qu'on désire passer réellement, ce qui est bien le cas du zero-knowledge. Qui nous dit qu'un nombre aléatoire est réellement aléatoire ? Généralement, le prouveur peut transmettre des informations autres que celles qu'on lui a demandées, notamment dans un protocole d'identification !

V.2.1 Exemple non informatique.

Considérons le protocole zero-knowledge à partir de l'isomorphisme de graphes. On suppose que les graphes sont représentés par des dessins dans le plan. Lorsqu'on construit un nouveau graphe à chaque round, on s'arrange pour que les arcs se croisent n fois (ou un nombre modulo précis, car les graphes ne sont généralement pas planaires). Ce nombre n est une information de nature très différente, transparente dans l'esprit de la vérification, mais qui permet de tricher ! Evidemment, d'un point de vue informatique, les croisements n'ont pas de sens.

Mais cet exemple est démonstratif. En fait, l'information est **résiduelle** ; elle n'est pas propre à la description mathématique du problème. On aurait pu tout aussi bien colorer le graphe par des couleurs différentes. Ces fraudes mettent en évidence la distance entre les protocoles théoriques et les applications réelles.

V.2.2 Problème du nombre aléatoire.

Dans presque tous les protocoles utilisant les grands nombres premiers, il faut générer aléatoirement un nombre u. Mais rien n'empêche de s'arranger pour qu'il ait un sens détourné sans que sa propriété aléatoire disparaisse.

Pour empêcher cela, la première idée consiste à éviter l'apport d'information résiduelle. On peut utiliser une fonction à sens unique injective, soit Ξ.

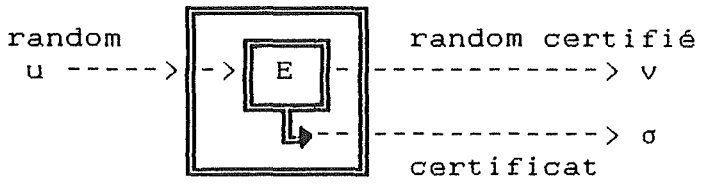
On choisit v tel que $u = \Xi(v)$. De cette manière, on ne peut plus statuer sur des propriétés de u. Car on ne peut pas construire u puis v.

Malheureusement, ce n'est pas simple car il faut vérifier la connaissance de v, sans donner v. Evidemment, sinon le subliminal peut être v ! Ce doit donc être effectué par un zero-knowledge. Et il faut que celui-ci ne possède pas de subliminal ! D'où danger de tourner en rond.

Ceci mène à plusieurs solutions proposées par J.J.Quisquater :

1) le nombre aléatoire est donné à la fois au prouveur et au vérificateur, par une autorité en laquelle il faut placer sa confiance.

2) on définit des **randoms certifiés** [Quisquater, 87] : on exploite l'idée développée précédemment, et qui consiste à envoyer v au lieu de u. Comme on l'a fait remarquer, la difficulté est de montrer que la fonction Ξ a réellement été utilisée. Pour réaliser cela, il faut passer par une certification non-falsifiable de carte, qui possède donc une zone tampon ("tamperproof device"). Par exemple :



3) On construit un zero-knowledge d'un nouveau type qui consiste en la limitation de l'usage de nombres aléatoires. Pour ce faire, on limite le nombre d'interactions dans le protocole à une seule interaction ! Cette idée a conduit L.Guillou et J.J.Quisquater au développement d'une nouvelle méthode d'identification/authentification en une passe.

V.2.3 Authentification en une passe.

La méthode [(Draft) Pour la science Juin 88] possède déjà un caractère très pratique en vue d'une application sur une carte.

Soit N , un entier codé sur 512 bits et composé de façon maintenant classique par deux grands nombres premiers. Soit P, un entier codé sur 16 bits (ou plus, si l'application demande un très haut niveau de sécurité), premier avec la fonction d'Euler $\Xi(N)$ [= (P-1)*(Q-1)] (ceci pour des raisons d'intraitabilité).

Chaque carte possède un identifiant bancaire I, codé sur 256 bits (numéro de compte, numéro de série de la puce, période de validité,...). A partir de I, on construit l'identifiant redondant J codé sur 512 bits (I redoublé, par exemple).

L'autorité bancaire va introduire une valeur d'authentification dans chaque carte durant la personnalisation. Soit B , codé par 512 bits également. B est calculé par l'autorité bancaire de la façon suivante :

$$J*B^F \bmod N = 1.$$

Pour retrouver B, il faut connaître les facteurs de N. Or ils sont en la possession de l'autorité bancaire uniquement. Si on ne connaît pas les facteurs à l'avance, factoriser un nombre de 512 bits est impossible dans un avenir prévisible; et retrouver B est impossible (Log Discret).

Quel est le protocole ?

A chaque demande d'authentification de la part du vérificateur au prouveur qui est la carte :

- la carte tire au hasard r , de 512 bits, et calcule la valeur de **test** $T = r^F \bmod N$.

- La carte montre I et T au vérificateur.

- Le vérificateur pose une question à la carte en lui donnant $0 < D < P-1$, D aléatoire.

- La carte calcule le **témoin** $t = r*B^D \bmod N$.

- La carte montre t au vérificateur.

- Le vérificateur vérifie que $T = J^D*t^F \bmod N$.

En effet, $T = J^D*(r*B^D)^F \bmod N$

$$T = r^F*(J*B^F)^D \bmod N$$
$$= 1.$$

Remarque importante :

La valeur de test T est en fait un random certifié par r et t.

1) T est aléatoire, puisque r l'est.

2) T n'a pas de subliminal, car pour construire t, il fallait connaître r. (Exactement comme précédemment : pour construire v, il fallait connaître u).

V.2.4 Variante de double authentification.

Dans le protocole précédent, il a été montré que le nombre aléatoire T du prouveur ne possède pas de subliminal. Par contre, il n'en est rien pour le nombre aléatoire D du vérificateur.

L'idée développée ici est une variante du point précédent, qui met à profit à la fois l'authentification et la suppression de subliminal.

En effet, pourquoi ne pas construire une double interaction où le nombre aléatoire D proviendrait de la sortie d'un schéma identique au point précédent ? De ce fait, D n'aurait plus de subliminal. De plus, et c'est finalement l'essentiel, ceci aurait pour avantage de construire un protocole d'identification **mutuelle en une passe** sans aucun apport de connaissance !

Il se pose un petit problème technique, en ce sens que le nombre aléatoire T est codé sur 512 bits, tandis que D est codé sur 16 bits. Qu'à cela ne tienne, il suffit de construire une fonction de réduction $\Gamma(X,P)$ qui permet de passer d'un nombre aléatoire X de 512 bits à un nombre aléatoire de 16 bits inférieur à P. C'est par exemple une troncature.

Soient J1,B1,P1,N1 définis comme précédemment pour le prouveur/vérificateur n°1. Et respectivement J2,B2,P2,N2 pour le prouveur/vérificateur n°2. On a donc :

P/V n°1		P/V n°2
$J1*B1^{P1} \bmod N1 = 1$		$J2*B2^{P2} \bmod N1 = 1$
random $r1$ ---▶ $T1 = r1^{P1} \bmod N1$		$T2 = r2^{P2} \bmod N2$ ◀--- random $r2$
	échange gradual release par exemple	
$T2,J2,N2,P2$	◀----------▶	$T1,J1,N1,P1$
	déf: $\Gamma(T2,P1) = D1$, $\Gamma(T1,P2) = D2$	
calcul : $t1 = r1*B1^{D1} \bmod N1$		calcul : $t2 = r2*B2^{D2} \bmod N2$
	échange gradual release par exemple	
$t2$	◀----------▶	$t1$
Vérification		Vérification
$T2 = J2^{D2}*t2^{P2} \bmod N2$		$T1 = J1^{D1}*t1^{P1} \bmod N1$.

La sortie T de l'un comme de l'autre, prenant le rôle de prouveur, sert comme entrée D (après troncature) de l'un comme de l'autre, prenant le rôle de vérificateur. Ensuite, il n'y a qu'à vérifier la correction de l'identification au moyen du témoin t.

V.2.5 Remarque.

Les deux protocoles des points V.2.3 et V.2.4 sont en fait de vrais protocoles zero-knowledge, mais d'un type nouveau. En effet, ils **ne se situent plus** dans une classe IP(k) (k>1), c'est-à-dire en temps que Preuves Interactives en k passes [cfr VI.1.4]. Cependant, le nombre aléatoire D réduit en quelque sorte toutes les passes à une seule. Le niveau de sécurité se reporte entièrement dans la fiabilité du schéma exponentiel.

CHAPITRE VI

QUALITE DE LA SECURITE.

VI.1 ALGORITHMES PROBABILISTES.

VI.1.1 Importance en cryptographie.

Les algorithmes probabilistes sont d'un grand intérêt et d'une grande actualité en cryptographie.

En effet, la cryptographie se base sur la théorie des nombres et plus particulièrement sur la difficulté de factoriser de grands nombres. Or, les algorithmes qui effectuent cela le plus rapidement à l'heure actuelle sont de type probabiliste, et les développements récents tendent à montrer qu'ils sont la meilleure façon d'aborder ce genre de problèmes.

Il est évident que le nombre de chiffres exigés dans les applications cryptographiques doit être augmenté si le temps de résolution de ces problèmes diminue. Heureusement, dans un avenir raisonnable, **ces problèmes restent complexes** ; c'est ce qu'on recherche en cryptographie. Le tout est de maintenir un écart suffisant entre le nombre de chiffres en limite supérieure des meilleurs algorithmes probabilistes (±300 bits actuellement) et le nombre de chiffres de l'application sécuritaire (±500 bits). En effet, la complexité est exponentielle; et plus les performances de ces algorithmes seront grandes, plus l'écart entre leur limite supérieure en nombre de chiffres et celle de l'application, paradoxalement, pourra diminuer. Bien sûr, plus on sera exigeant au niveau sécurité, plus l'écart sera choisi grand.

Il est donc essentiel d'étudier de près les développements récents des algorithmes probabilistes.

VI.1.2 Catégories d'algorithmes probabilistes.

Il existe principalement quatre types d'algorithmes probabilistes :

1) Algorithmes numériques donnant des solutions approximatives par intervalles de confiance. Par exemple, un des algorithmes consiste à compter jusqu'à 2^n avec comme place mémoire log(n) bits, et ce à un facteur 2 près. Cette catégorie nous intéresse peu et ne sera pas développée; néanmoins, des perspectives s'ouvrent quant aux "grillages" de bits de cartes à puce...

2) Algorithmes Monte-Carlo.

3) Algorithmes Las Végas.

4) Algorithmes Sherwood.

Pour tous ces algorithmes, à chaque étape à partir de laquelle on doit décider quel chemin est le meilleur, on conseillera de "foncer" plutôt que de réfléchir longtemps. Bien sûr cet a priori peut amener à de mauvais choix. Le fait de prendre une décision rapidement peut ne jamais donner de réponse. Mais ce n'est pas grave : si on est perdu, on arrête et on recommence, en oubliant tout ce qu'on vient de faire !

En fait, pour les types 2), 3) et 4), on parle de pseudo-aléatoire car le résultat est totalement **déterministe**.

Monte-Carlo

Les algorithmes Monte-Carlo peuvent se tromper, mais possèdent l'avantage de donner toujours une réponse. En général, on ne peut pas déterminer efficacement si la réponse obtenue est bonne.

Las Végas

Les algorithmes Las Végas possèdent l'avantage de ne jamais donner de réponse inexacte. Mais parfois, ils ne trouvent pas de réponse du tout, quelle que soit l'entrée.

Les méthodes de Monte-Carlo et de Las Végas peuvent évidemment se réexécuter autant de fois que bon nous semble. Il y a cependant une différence fondamentale entre les conclusions de ces répétitions propres à chaque algorithme.

Si Monte-Carlo donne très souvent la même réponse, on peut espérer qu'elle soit correcte.

Si Las Végas ne donne jamais de réponse, on peut supposer que l'espérance mathématique du temps d'arrêt est anormalement faible du point de vue d'un comportement naturel, si la réponse (en temps que vérification d'un énoncé : par exemple "ce nombre est-il premier ?") est correcte.

Sherwood

Les algorithmes de ce type donnent toujours la bonne réponse. Ils sont généralement des boucliers contre des distributions biaisées des entrées. Pour ce faire, ils se basent sur l'algorithme déterministe correspondant, qui est efficace en moyenne, mais dont l'écart-type est important. Le but ici est de réduire l'écart-type sans changer l'efficacité moyenne. Ceci est un principe ressemblant à du préconditionnement.

VI.1.3 Analyse cas par cas.

Monte-Carlo

Un algorithme est dit **P-correct** si il donne une solution avec probabilité ≥ P. Si l'**avantage**, défini comme étant P-1/2, est positif, on peut **amplifier** de manière exponentielle la probabilité de solution correcte.

Ainsi, on peut construire un algorithme $(1-\delta)$-correct à partir d'un algorithme $(1/2+\epsilon)$-correct. De tels algorithmes existent. L'algorithme de Rabin basé sur les nombres de Kermaker (fortement pseudo-premiers) [Rabin, 76] est de ce type. Il permet de savoir si un nombre est factorisable ou non. Cet algorithme est 3/4-correct.

Les méthodes Monte-Carlo sont également souvent utilisées pour mesurer des aires de polygones ou intégrales complexes. Le principe de résolution est un principe de comptage sur projection de grilles-pavés dont les points sont équiprobables. Malheureusement ces méthodes sont souvent instables.

Las Végas

Voici comment se présente un algorithme de ce type :

PROCEDURE LV(X , Var Y , Var Succès)

Au Retour : Succès = vrai => Y est solution de X.

Succès = faux => pas de chance !

Le but est de borner l'espérance mathématique du temps moyen avant le premier succès (soit t).

Soit p = probabilité de succès.

Soit s = temps d'une exécution moyenne si elle aboutit à un succès.

Soit e = temps d'une exécution moyenne si elle aboutit à un échec.

On a logiquement : $t = p*s + (1 - p)*(e + t)$ d'où on tire

$$t = s + ((1 - p)/p)*e.$$

On a généralement que quand s augmente, p augmente et e diminue. D'où il y a une valeur optimale.

Exemple des 8 reines.

Le problème consiste à placer 8 reines sur un échiquier de 64 cases de telle sorte qu'aucune d'elles ne soit en prise au sens du jeu d'échec.

On place $1 \leq k \leq 8$ reines au hasard sur l'échiquier, ligne après ligne. Toutefois, sur une ligne, on ne considère comme positions possibles, que les positions qui ne sont pas dans le champ des reines déjà placées aux lignes précédentes. On choisit au hasard parmi les positions valides de cette ligne. Si ça rate, c'est à dire qu'une des lignes n'admet plus de position valide, on recommence à zéro, et ce jusqu'à un succès.

On obtient par cette méthode un résultat étonnant : cet algorithme réussit en moyenne après 12,93 essais !

On constate que, en moyenne, c'est déjà plus rentable que le backtracking classique (2x plus rapide).

Mais on peut faire beaucoup mieux en recherchant l'optimum pour t. Si on place 3 reines de façon aléatoire, et qu'on poursuit par du backtracking sur les 5 dernières, sans toutefois revenir en arrière sur les 3 premières rangées (dans ce cas, on considère que c'est un échec), on obtient un algorithme de Las Végas très efficace. En effet, la probabilité de réussir sur un essai est d'environ une chance sur deux !

G.Brassard a montré que si on se ramène au problème qui consiste à placer 20 reines sur un échiquier de 200 cases, le backtracking sur un AppleII prend 2 heures et qu'en plaçant 5 reines de façon aléatoire sur les 5 premières lignes et en continuant par le même programme de backtracking, on trouve en moyenne la solution en moins de 6 minutes.

Pour l'instant on ne sait pas déterminer par calcul les valeurs optimales du nombre de reines à placer de façon aléatoire en fonction du nombre total de reines.

Il est possible dans certains cas de passer de Monte-Carlo à Las Végas.

Regardons de plus près la façon dont on peut améliorer par un Las Végas l'algorithme de test de primalité de Rabin qui est du type Monte-Carlo. On utilise l'algorithme GKAH qui est du type Las Végas [Goldwasser & Killian, 86; Adleman & Huan, 87].

```
LV_PREMIER( N , Var Réponse, Var Succès )

  SI Rabin (N) alors (avec proba = 3/4)
                    Succès = Vrai
                    Réponse = Faux
  SINON SI GKAH (N)
            alors (avec proba = 1/2)
                    Succès = Vrai
                    Réponse = Vrai
        SINON       Succès = Faux.
```

Ce qui ne veut pas dire que la probabilité d'échec sera de 1/4 * 1/2 car il n'y a pas indépendance. Néanmoins, on a considérablement augmenté la probabilité d'obtenir une réponse.

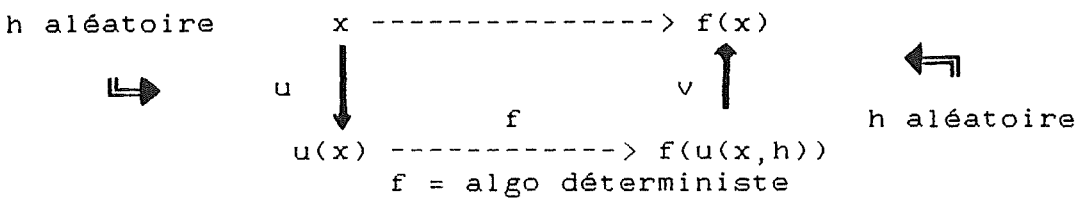
Sherwood

Tri d'un vecteur.

Considérons le problème déterministe de tri d'un vecteur. On pose le problème suivant d'élimination des exemplaires de pire cas (exemple repris d'une conférence de G.Brassard) : il faut trouver l'algorithme qui trie le plus rapidement le vecteur, étant donné qu'on peut choisir le vecteur que l'on veut (appelé **exemplaire**), en fonction de l'algorithme lui-même.

Pour bien comprendre le problème, si l'algorithme utilisé est Quicksort, un exemplaire trié sera un choix néfaste. En effet, on sait que dans ce cas, bien que le temps moyen est de o(n*log(n)), pour cet exemplaire, il est de $o(n^2)$. Donc, pour cet exemplaire, Shellsort ou Heapsort est plus rapide.

On s'intéresse à l'algorithme le plus rapide a priori. L'idée est d'effectuer un prétraitement stochastique:



L'idée est de prétraiter de façon aléatoire l'échantillon. De ce fait, on ne peut statuer sur la structure de l'algorithme, car l'entrée de l'algorithme déterministe est aléatoire !

Donc, si on permute de façon aléatoire l'exemplaire choisi, il n'y a plus de détermination explicite d'un exemplaire de pire cas en fonction de cet algorithme.

Hashing.

Dans la même lignée, les derniers compilateurs d'IBM utilisent des fonctions de hashing aléatoires; c'est-à-dire qu'elles se basent sur des classes, dites **universelles**$_{\geq}$, de fonctions surjectives à risque de collision faible (à distribution équilibrée).

En effet, si le compilateur utilise toujours la même fontion de hashing pour accéder rapidement aux adresses de la table de variables, il se peut que par malchance le problème soit très mal conditionné. De nombreuses variables se trouvent en collision et le comportement du programme lors de l'exécution peut être mauvais. Mais ce genre de comportement ne permet pas de décider où se situe le problème. Tandis que si la fonction change à chaque compilation, on recompile, et avec très grande probabilité le problème disparaît.

Une simple compilation permet de décider si le comportement mauvais est dû au hashing ou pas. On comprend que ce genre de compilateur arrange IBM.

VI.1.4 Théorie de la complexité et algorithmes probabilistes.

Considérons la classe des preuves interactives du type zero-knowledge. Elle s'appelle IP(k) où $k \geq 2$ représente le nombre de rounds. En fait AM = IP(k) , où AM représente une généralisation de NP aux algorithmes probabilistes.

RP est la classe des algorithmes de type Monte-Carlo.

ZPP est la classe des algorithmes de type Las Végas.

BPP est la classe des algorithmes de type Sherwood.

NBPP, défini de manière analogue à NP, représente en fait la classe MA de Babai [Babai, 85] (Arthur Merlin games).

On a en plus que MA est inclus dans AM.

MA est en fait une classe plus naturelle pour des preuves "pratiques". Pour plus de précisions, on consultera [Brassard, Chaum & Crépeau, 87, pp.12-13].

VI.1.5 Point de vue philosophique.

On est amené à penser qu'une preuve déterministe ne vaut pas plus et même souvent moins -en pratique- qu'une preuve probabiliste.

En effet, si la preuve déterministe nécessite une démonstration kilométrique, la probabilité d'erreur va devenir plus grande que celle donnée par l'exécution d'un algorithme probabiliste. C'est tellement étonnant que, bien que la probabilité d'une erreur engendrée par l'ordinateur lui-même est faible, ce risque peut devenir non négligeable si trois années sont nécéssaires pour exécuter le programme de façon séquentielle. A ce propos, si une exécution de cette longueur peut paraître non raisonnable, on peut exécuter en parallèle et de plus, dans un tel système, la probabilité d'erreur logicielle d'exploitation n'est plus négligeable.

VI.2 SECURITE AU NIVEAU DE L'IMPLEMENTATION DE LA PROCEDURE D'IDENTIFICATION.

Après avoir montré quelles étaient les limites mathématiques dérivées de la théorie des nombres et plus particulièrement du fait que P apparaît -dans la limite des connaissances actuelles- différent de NP, il est possible d'obtenir une qualité de sécurité suffisante du moins en théorie.

VI.2.1 Fraudes possibles.

Mais, à côté de cela, subsiste nombre de problèmes dépendant de chaque application. Des fraudes sont possibles même si chaque individu possède une description physique unique qu'on peut vérifier de façon certaine et que la vérification ne peut être fraudée.

La fraude consistant à copier le SID et à le garder en vue d'une utilisation ultérieure est possible pour les systèmes à clés publiques. Elle semble résolue pour les zero-knowledge. On peut montrer que ce n'est que partiellement vrai. En effet, quatre fraudes majeures sont possibles [Bengio, Brassard & al., 87] :

1) le propriétaire de la carte peut volontairement dévoiler son SID à quelqu'un d'autre.

2) le SID peut être utilisé par un individu autre que le propriétaire du SID, et ceci avec ou sans l'accord du propriétaire.

3) celui qui possède la carte peut la LOUER; peu importe à ce moment qu'il sache ou non lire ou connaître son propre SID.

4) la carte peut être volée ou perdue, et utilisée contre le gré de son propriétaire.

Il peut être intéressant de copier sa propre carte si le nombre d'accès est limité (par un certain nombre de bits de la carte "grillés" à chaque utilisation) ou encore pour déléguer ses pouvoirs, par exemple si on est trop occupé.

Le restaurant de la mafia.

Le SID de la carte peut être utilisé sans que le propriétaire ne le sache, et ce par une variante de l'attaque par transparence [cfr III.4.3.2].

Soit A, le propriétaire de la carte. B est le gérant d'un restaurant faisant partie de la mafia. C est un membre du même gang de mafia que B. D est un bijoutier. A et D ne sont pas conscients de la fraude. La carte permet à A de confirmer son identité lors du payement par chèque du repas pris dans le restaurant de B. A ce moment, le lecteur truqué de B est relié par radio à la carte truquée de C; ce dernier, entré préalablement chez D pour acheter un bijou, place sa carte dans le lecteur de D lors de la vérification d'identité de A (en fait C). La fraude peut commencer.

B va faire croire à A qu'il pose des questions à la carte de A pour l'identifier. Mais en fait C se fait passer pour A. Le lecteur de D pose des questions à la fausse carte de C concernant l'identité de la personne à qui il croit avoir affaire, c'est-à-dire A. La carte de C communique les questions par radio au lecteur de B qui les pose à A. La carte de A répond au lecteur de B, qui communique la réponse par radio à la carte de C, et cette dernière répète la réponse à D. B et C ne font que transmettre fidèlement les questions et réponses de A et D.

Dans d'autres contextes, d'autres variantes frauduleuses peuvent être mises en évidence. Bien sûr, ce ne sont que des illustrations de protocoles dont l'intérêt doit être resitué dans un **contexte informatique**.

Terrorisme.

Soit A, qui, moyennant payement, veut aider B à entrer dans un pays X. A possède un passeport valable pour le pays X. Chaque fois que B doit prouver son identité à C (qui est douanier, par exemple), B se place entre A et C, et la vérification du passeport se fait "au travers" de B.

Mais si B est un terroriste, la responsabilité de ses actes en incombe à A !

Un coup fourré est toujours possible, comme le montre cet exemple extrême :

A loue un passeport à B. Mais en fait, ce n'est pas pour aider B, mais pour que A puisse se forger un parfait alibi en la personne de B pour commettre un acte de terrorisme. Mais c'est à double tranchant si B commet lui-même un acte de terrorisme (qu'il peut même signer !).

Conclusion:

De nombreuses fraudes ne sont pas évitées par les techniques de zero-knowledge élémentaires.

VI.2.2 Remèdes.

Le remède idéal contre les fraudes décrites précédemment consiste en l'adéquation parfaite entre la personne et le SID. Ceci suppose donc que le SID se base sur des caractéristiques **physiques** de la personne.

Mais même à ce niveau, si le fraudeur en a le courage, il peut forcer la personne à agir comme il l'entend voire couper les mains de la personne pour utiliser ses empreintes digitales ou la tuer dans le cas d'identification du visage (empreintes rétiniennes,...).

VI.2.2.1 Empêcher la copie.

Personne ne doit pouvoir lire le SID. Pour ce faire, le SID est placé dans une zone tampon de la carte, qui est une sorte de boîte étanche contrôlée par un microprocesseur entièrement indépendant de l'extérieur. Il faut être extrêmement prudent afin d'empêcher le propriétaire lui-même de calculer son SID. Attention : même si physiquement le SID est inaccessible, une déduction après calcul peut être possible. Même si trois ans après l'utilisation de la carte, on a réussi pendant ce temps à exploiter les calculs résultant de l'identification, la connaissance du SID peut encore être pertinente. Tout dépend du type d'application.

VI.2.2.2 Empêcher l'attaque par transparence.

Il faut conduire le protocole dans une boîte neutre synchronisée et contrôlée par un garde, et dans laquelle le vérificateur a placé sa carte d'interrogation et le prouveur sa carte d'identité. Le rôle de la boîte neutre est d'empêcher toute fraude venant soit du prouveur soit du vérificateur. Mais il n'est pas du tout évident d'empêcher la communication avec l'extérieur.

Pour être sûr que les deux cartes soient bien placées, et qu'on puisse effectivement commencer la transaction, il faut établir un TIMING géré par le garde. La réalisation pratique passe par la division en 2 cages dites "de Faraday" dont tous les messages entrant et sortant sont signés.

Pour plus de précisions, on consultera plus particulièrement [Bengio, Brassard & al., 87, pp.12-18]. Cet article développe la complexité de réalisation de protocoles d'identifications sûrs étant donné la diversité des attaques possibles.

VI.2.2.3 Empêcher la location.

La seule possibilité est d'obliger le propriétaire à garder sa carte sur lui : on le force à la montrer très souvent. C'est-à-dire que la carte devient un moyen de certifier l'identité d'une personne. Des chèques peuvent par exemple être certifiés par carte à puce [Cfr Sc&Vie Avril 88].

La carte devient véritablement une carte d'identité non-falsifiable et à usage multiple. Le préjudice lors de sa perte devient énorme. L'intérêt de sa falsification devient énorme pour des organisations d'espionnage militaire qui disposent d'attaques qui peuvent approcher la hauteur des précautions prises.

VI.2.2.4 Empêcher la perte et le vol.

Etant donné que le contrôle physique de haute sécurité coûte cher, on remplace la description physique par un Numéro d'Identification Personnel (PIN) ou numéro de code de type mot de passe. Mais cela ne résout rien si on ne prend pas un minimum de sécurité lors de la communication du PIN par le propriétaire au lecteur de la carte.

VI.2.3 Conclusion.

Les preuves mathématiques existent pour les protocoles mathématiques mais il n'en est rien pour les protocoles reliés à l'implémentation. La prudence est de rigueur.

CHAPITRE VII

IMPLEMENTATION DE "ZERO-KNOWLEDGE".

VII.1 DIFFICULTES D'IMPLEMENTATION.

Les difficultés qui surgissent lors de l'implémentation sont de deux types. Il s'agit d'abord de difficultés liées aux **calculs** de grands nombres et aux générateurs aléatoires qui nécéssitent la mise en oeuvre de routines efficaces. Il s'agit ensuite de difficultés liées au dialogue entre le prouveur, le vérificateur et bien souvent une autorité. La gestion de ce dialogue pose des problèmes d'**interactivité**, de priorité et surtout de sécurité.

L'application décrite dans ce mémoire est un protocole zero-knowledge basé sur les résidus quadratiques [Cfr IV.3.3].

Dans cette application, on essaye de se sensibiliser aux grands nombres afin de se rendre compte de la signification réelle d'un exemple de fonction trappe.

On décrit aussi des zones de contrôle **simulées**. En effet, une application de ce type concerne évidemment deux processus concurrents : le prouveur et le vérificateur. Il aurait été plus réaliste de construire l'application en environnement partagé. Mais il s'agit d'un exemple **démonstratif**. Néanmoins, la grandeur des chiffres utilisés est **réaliste**.

Le programme réalisé est écrit en langage C. Car, bien que non élémentaires, les méthodes employées pour faire la multiplication ou l'exponentielle dans un espace modulo par exemple, ne sont pas les meilleures utilisées dans les applications cryptographiques, comme le RSA. D'ailleurs, ces calculs devraient s'effectuer de façon hardware, dans une puce microprogrammée ou mieux, câblée. Il nous faut donc un langage dont l'exécution est rapide.

VII.1.1 Calculs.

On a besoin d'un "package" de gestion de grands nombres entiers. La première remarque concerne les formats de ces nombres : il s'agit de formats **multi-longueurs**.

Les multi-longueurs sont en fait des nombres exprimés dans des bases élevées. Dans le programme considéré, la base (définie dans la variable BASE) est 10000, ce qui est intéressant car toute base puissance de 10 se lit facilement en décimal de gauche à droite.

L'exponentielle modulo ne doit pas s'effectuer dans le sens : d'abord l'exponentielle et ensuite le modulo. Tout comme pour la multiplication, il existe des méthodes basées sur le principe de "DIVISER POUR REGNER", du type exponentiation indienne. En effet : $w^{16} = (((w^2)^2)^2*w)^2$. On passe de 17 multiplications pour la méthode élémentaire qui consiste à multiplier w 17 fois; à multiplier 5 fois seulement, si on considère qu'une mise au carré compte pour une multiplication. Il faut utiliser au maximum les calculs intermédiaires. De même, pour voir si deux nombres sont premiers entre eux, on utilisera l'algorithme d'Euclide plutôt que bêtement factoriser et constater qu'il n'y a pas de facteurs communs.

Par les techniques traditionnelles, la multiplication [$o(n^2)$] est beaucoup plus lourde que l'addition [$o(n)$]. En 1971, A. Schönhage et V. Strassen montrèrent qu'en théorie, la complexité de la multiplication était à peine supérieure à celle de l'addition. Pour tendre vers cette réduction, il est commode d'utiliser la transformée de Fourier rapide (FFT). La multiplication de deux grands nombres par FFT orchestre si rigoureusement les calculs intermédiaires entre chiffres, qu'elle élimine drastiquement toute répétition [Pour la Science, Avril 88 p.42].

Il est à remarquer que l'extraction de racine, la division et la mise au carré sont de même complexité que la multiplication (on démontre par réduction linéaire).

VII.1.2 Interactivité & arbitrage.

L'interactivité, essentielle dans ce genre de protocoles, doit s'opérer de façon sûre. On se borne à la simulation : les calculs secrets propres à chaque parti sont effectués dans des zones protégées. Le programme se limite à la description de ces zones, représentées sous forme transactionnelle. Elles se décomposent en trois types :

1° Zone de présélection du vérificateur et zone pseudo-commune.

Il s'agit de zones dans lesquelles les deux partis se mettent préalablement d'accord sur le mode d'utilisation de(s) générateur(s) aléatoire(s), chacun étant secret pour l'autre parti. Les différents paramètres utilisés sont distincts dans une application réelle. Néanmoins, pour des raisons de simplicité, le générateur du programme est commun [cfr VII.3].

De plus, le nombre N = p*q, essentiel dans l'application, sera communiqué.

En fait, cette zone sera sous le contrôle d'une autorité qui **arbitrera** les rounds. Cette autorité aura forgé le nombre N; elle sera donc la seule à connaître les facteurs p et q (elle peut même les avoir oubliés !).

2° Zone de présélection du prouveur.

La zone de présélection du prouveur consiste en le choix d'un mot de passe w et le calcul de son certificat public y. Ce certificat sera placé dans l'annuaire, comme identification publique (ID) de w (SID).

Il est évident que, pendant cette phase initiale, le vérificateur doit être convaincu que y est bien le certificat du prouveur. L'avantage réside ici dans le fait que c'est effectué **une fois pour toutes**.

Mais ce protocole ne décrit pas comment s'assurer de la correspondance correcte entre y et le prouveur.

3° Zone du prouveur et zone du vérificateur.

Il s'agit de zones dans lesquelles chaque parti doit être à même d'effectuer de façon privée les calculs nécessaires à l'identification. C'est particulièrement vrai pour le prouveur, car il effectue des calculs avec son mot de passe.

VII.2 PRESENTATION DU PROGRAMME.

VII.2.1 Description.

Le programme est l'implémentation du protocole zero-knowledge des résidus quadratiques (ou protocole de base de Fiat-Shamir) développé au point IV.3.3. Les symboles et variables utilisés sont les mêmes que ceux du protocole théorique. La structure d'interaction entre le prouveur et le vérificateur est calquée sur le programme en BASIC dans la revue BYTE [BYTE, Oct 87].

Les procédures de manipulation de grands entiers sont décrites en PASCAL dans [Riesel, 85, Appendix A]. Les primitives sont : addition/soustraction, multiplication, exponentiation dans un espace discret, division entière, vérification si deux nombres sont premiers entre eux. Pour des raisons de performances, elles ont été réécrites en C, en gardant les mêmes noms de variables. Les procédures d'entrée/sortie ont été entièrement modifiées pour permettre des saisies interactives ou par fichier.

Le générateur aléatoire utilisé, est expliqué au point VII.3. Le programme se trouve en annexe.

VII.2.2 Résultats.

On a considéré un exemple dont la dimension des entiers correspond (est même supérieure) à celle des applications réelles.

On a pris $p = 2^{521} - 1$ et $q = 2^{607} - 1$, qui sont deux grands nombres premiers. $N = p*q$ est un nombre de 342 chiffres, ce qui est supérieur à la sécurité de la plupart des applications.

Le temps d'exécution est de l'ordre de 2 minutes CPU, ce qui est évidemment inacceptable. Cependant, il faut se rendre compte que la méthode de gestion des grands nombres utilisée est loin d'être la plus efficace et que l'ordre des chiffres est un peu trop élevé. De plus, le protocole pourrait être microprogrammé ou cablé.

A chaque round, le vérificateur est convaincu, ce qui laisse d'ailleurs supposer que le programme est correct (surtout si on manipule de si grands nombres). Le nombre de rounds étant limité à 10, cela laisse théoriquement au prouveur moins d'une chance sur 1000 de tricher.

Néanmoins, suite aux travaux récents, ce protocole n'aura sans doute pas beaucoup d'avenir pratique, si ce n'est le protocole opérationnel de Fiat-Shamir opéré en parallèle [Fiat & Shamir, CRYPTO'86].

Le fichier résultat se trouve en annexe.

VII.3 GENERATEURS ALEATOIRES OPERATIONNELS.

VII.3.1 Générateurs de simulation.

Il s'agit ici d'un procédé de génération arithmétique, qui est un algorithme créant une suite récurrente de nombres entiers et prenant les chiffres de la suite, comme chiffres au hasard. En raison de la nature récurrente de la suite, il est évidemment impossible d'obtenir des chiffres correspondant à des résultats d'expériences mutuellement indépendantes. Cependant, ces procédés engendrent des suites de nombres que l'on peut, en pratique, considérer comme statistiquement indépendants. Les nombres engendrés sont dits pseudo-aléatoires. Les procédés s'appellent générateurs, ils évitent l'emploi de tables colossales.

Pour l'application décrite au point précédent, comme dans tous les protocoles zero-knowledge, un générateur aléatoire est indispensable. Une méthode congruentielle linéaire de type $X(n+1) = a * X(n) + c \pmod M$ a été choisie. On conseille ici : M est un grand nombre premier -de 387 chiffres ($2^{1279} - 1 > N$)- , le facteur multiplicatif a est de l'ordre de la racine de M, le facteur additif c est de l'ordre de $0.2 * M$ [Knuth, 69]. Mais ceci n'a pas été vérifié expérimentalement. On peut espérer une période de l'ordre d'un nombre ayant 350 chiffres. La semence X(0) et les facteurs ont été choisis en introduisant une suite quelconque de nombres. Attention toutefois, les chiffres inférieurs ne sont pas aléatoires. Si on désire un bit aléatoire, considérer par exemple : 0, si le quatrième chiffre à partir de la droite est inférieur à 5; 1, sinon.

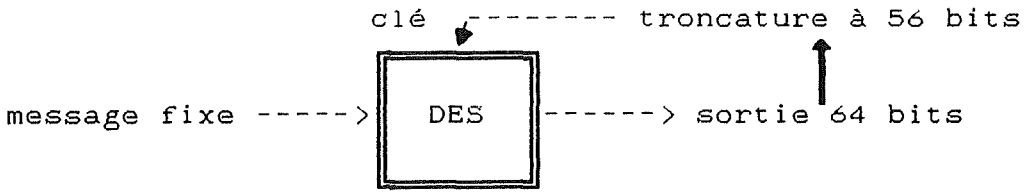
VII.3.2 Générateurs cryptographiques.

La méthode précédente est relativement efficace d'un point de vue simulation, mais probablement mauvaise d'un point de vue cryptographique (du moins l'approche sécuritaire est nulle).

L'exigence cryptographique est le fait qu'on ne puisse pas prévoir le nombre aléatoire suivant en connaissant les précédents. Ce n'est pas nécessaire en simulation, où on exige seulement que la suite de nombres ne soit pas biaisée. La présence de petits cycles est une bête noire.

En 1986, il a été montré que le comportement du taux de distribution de cycles dans les fonctions aléatoires est étonnant : la probabilité qu'une fonction possède une semence X(0) qui engendre un pourcentage efficace de noeuds X(n) avant de cycler est faible. En effet, J.-J. Quisquater a effectué des expériences à partir du DES qui ont montré que 3% du nombre total de noeuds, pris comme semences, aboutissaient à un cycle [fig VII.1] de petite taille (2^{16}) [Quisquater, 86].

La fonction utilisée se base sur le DES; étant donné une clé aléatoire et un message fixe, la sortie du DES est utilisée par la suite comme nouvelle clé, après troncature.



Remarquons enfin que cette fonction n'est cryptographiquement pas bonne, car on peut retrouver les valeurs en utilisant le DES^{-1}.

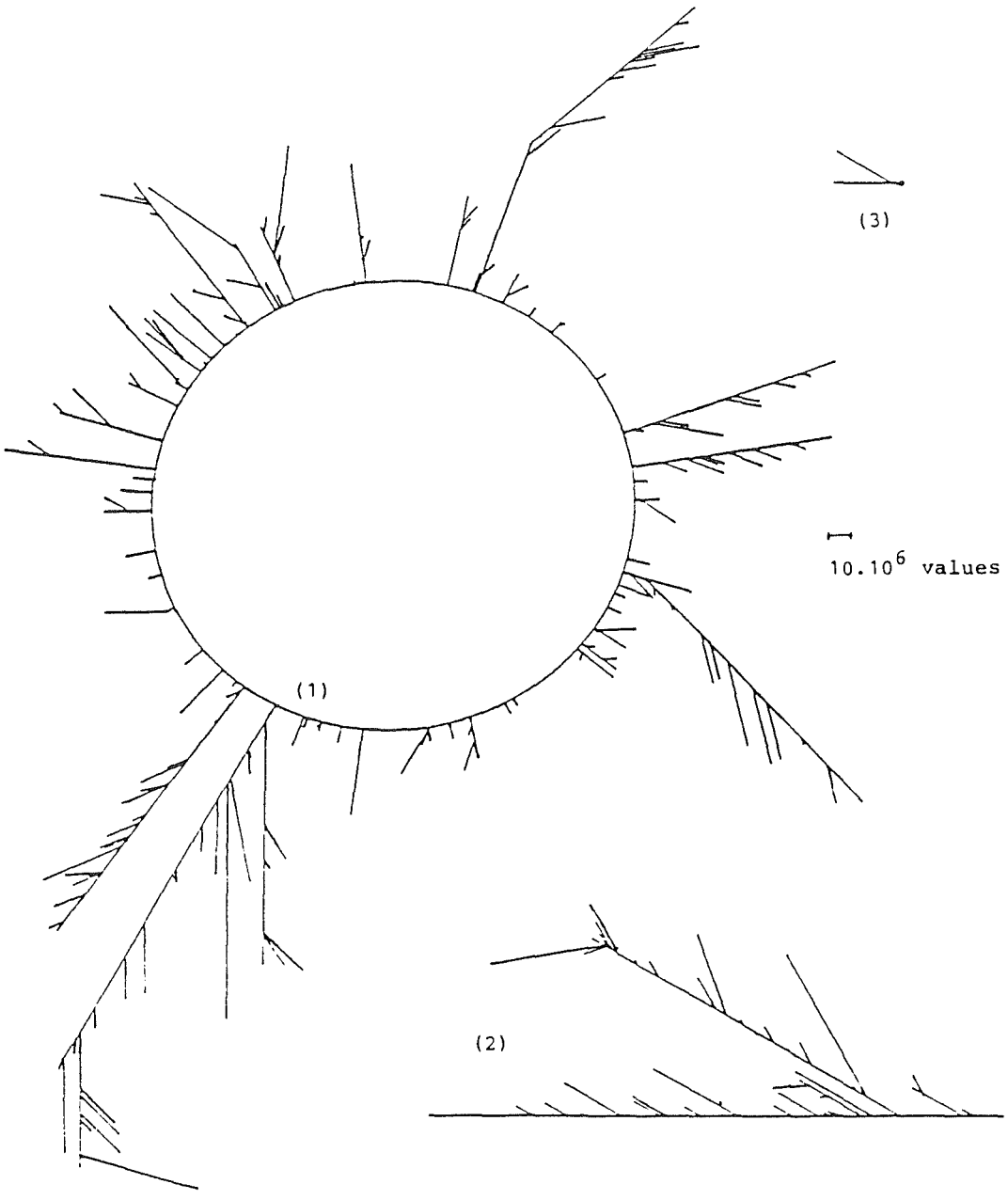


FIG VII.1 TEST DE CYCLES BASE SUR D.E.S. (2 EST TRES PETIT!).

CHAPITRE VIII

CONCLUSION.

Ce mémoire s'est intéressé à l'identification d'objets informatiques, donc décrits au moyen de bits et pour lesquels il n'existe pas d'unique description. En effet, rien ne ressemble plus à une chaîne de bits qu'un autre chaîne de bits dont la représentation est identique. Dès lors, éviter toutes les fraudes est impossible notamment à cause de l'attaque par transparence, qui consiste à agir en temps qu'intermédiaire en retransmettant fidèlement le dialogue d'un protocole d'identification entre deux partis.

Néanmoins, l'avenir de l'utilisation de zero-knowledge dans les protocoles d'identification est prometteur suite au concept essentiel d'**apport de connaissance nul** dans un dialogue interactif. En effet, ceci réduit les fraudes car le fraudeur n'acquiert pas d'information pouvant amener à "casser" le protocole.

Il ressort de ce mémoire qu'afin de bénéficier des propriétés paradoxales mais efficaces des protocoles zero-knowledge, il ne faut pas négliger les aspects sécuritaires au niveau de l'implémentation. En effet, un protocole théorique s'adapte difficilement à une implémentation rigoureuse. Les travaux récents s'approchent à grands pas de solutions implémentables sur carte à microprocesseur. Mais la prudence se doit d'être extrême car les pièges sont fréquents face à la diversité des fraudes possibles.

Les applications sont militaires ou privées : depuis les cartes bancaires jusqu'à la vérification de désarmement nucléaire mutuel en passant par toutes les variétés de contrôle d'accès.

Cependant, il est à déplorer que **tous** les protocoles cryptographiques se basent sur la théorie des nombres. Les algorithmes de factorisation de grands nombres se heurtent aux

limites de la théorie des nombres premiers. La difficulté intrinsèque de résolution du problème de la factorisation ou autre problème équivalent, est mise en évidence par la théorie de la complexité, même si on tient compte des progrès originaux des meilleurs algorithmes probabilistes.

Ce mémoire a proposé et développé de nombreuses nouvelles pistes exploitées notamment par Gilles Brassard, Ivan Damgård et Jean-Jacques Quisquater.

Il constitue une synthèse et une structuration des articles dont l'approche est difficile parce que la cryptographie est un carrefour de théories touchant à des domaines aussi divers que la théorie des nombres, la théorie de l'information, les probabilités, la théorie de la complexité ... De plus, il aborde le terrain riche mais mouvant de la recherche : les analyses globales concernant le sujet sont encore rares. Heureusement, les nombreux articles m'ont permis de dégager, je crois, un aspect essentiel de la cryptographie moderne : les protocoles zero-knowledge.

Dans un monde où se côtoient des concepts aussi étranges que zero-knowledge, blob, canal subliminal et transfert équivoque, c'est non seulement une nouvelle jeunesse qui se dégage de la science cryptographique (qui est aussi un art), mais surtout une nouvelle philosophie.

ANNEXE

******

programme et résultats.

Programme implémentant un
protocole "zero-knowledge"
basé sur les résidus quadratiques
[cfr IV.3.3 & VII.2].

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
#include <math.h>
#define sqrtb 100
#define vsize 1508
#define lvsize 3016
typedef int vector[vsize];
typedef char strng[50];

int base,b1,b101;

vector sem,asem,csem,pri;/* *sem pseudo common because random are
>< */
vector one,two,pq; /* p and q are Verifier property */
strng txt;
/*-------------------------------------------------------------*/
main()
{
vector w,u,v; /* Prover property*/
int round,i,prob,PROBOK,check;
vector choice,sort,aux,y,z; /*common variables*/
base=sqrtb*sqrtb;
b1=base-1;
b101=base / 10 -1;
printf("\nBase=10^%d\n",(int) floor(0.434294481*log((double)
base)+0.5));
/*~~~~~~PRESELECTED VERIFIER ZONE~~~~~*/
prob=1;
PROBOK=1024;
one[0]=1;
one[1]=1;
two[0]=1;
two[1]=2;
strcpy(txt,"f1279.dat");
fput_in(txt,pri);

strcpy(txt,"pq.dat"); /* is supposed to be given by the Verifier
*/
/*~~~END OF PRESELECTED VERIFIER ZONE~~~~~~~*/
fput_in(txt,pq);
/*~~~~PSEUDO COMMON ZONE (RANDOM)~~~~~~~*/
strcpy(txt,"fasem.dat");
fput_in(txt,asem);
strcpy(txt,"fsem.dat");
fput_in(txt,sem);
strcpy(txt,"fcsem.dat");
fput_in(txt,csem);
/*~~~~~~END OF PSEUDO COMMON ZONE (RANDOM)~~~~*/
/*~~~~~~PRESELECTED PROVER ZONE~~~~~~~~*/
select(w); /* password */ /* known by the Prover */
strcpy(txt,"\nPassword is:");
put_out(txt,w);
quadres(w,y);
strcpy(txt,"\nProver has given y:");
/*~~END OF PRESELECTED PROVER ZONE~~*/
```

```
put_out(txt,y);
round=0;
m1: /*ROUND*/
printf("ROUND %d\n",round);
round++;
/*~~~~~~~~~PROVER ZONE~~~~~~~~~*/
select(u);
quadres(u,z);   /* u and z known by the Prover */
strcpy(txt,"\nProver gives z:");
/*~~~~~~~END OF PROVER ZONE~~~~~~~*/
put_out(txt,z);
/*~~~~~~~~VERIFIER ZONE~~~~~~~~~*/
randbit(choice); /*random of verifier*/
strcpy(txt,"\nVerifier chooses:");
/*~~~~~~~~END OF VERIFIER ZONE~~~~~~~~*/
put_out(txt,choice);
/*~~~~~~~~~~PROVER ZONE~~~~~~~~~~*/
if (choice[1]) {
  /* choice=1 => Prover give v = (u*w) mod pq */
  strcpy(txt,"\nProver gives v:");
  mul(u,w,v);
  apowerb(v,one,pq,v);
  let(sort,v); }
else {
  /* choice=0 => Prover gives u = square root of given z */
  strcpy(txt,"\nProver gives u:");
  let(sort,u); }
/*~~~~END OF PROVER ZONE~~~~~~~~*/
put_out(txt,sort);
/*~~~~~~~VERIFIER ZONE~~~~~~~*/
/*checking*/
check=1;
if (choice[1]) {
  /* choice=1 => Verifier checks v^2 = (u*w)^2 = (z*y) (mod pq) */
  apowerb(sort,two,pq,sort);
  mul(z,y,aux);
  apowerb(aux,one,pq,aux);
  for (i=0;i<=sort[0];i++) if (sort[i]!=aux[i]) {
                              check=0;
                              break; }}
else {
  /* choice=0 => Verifier checks u^2 = z (mod pq) */
  apowerb(sort,two,pq,sort);
  for (i=0;i<=sort[0];i++) if (sort[i]!=z[i]) {
                              check=0;
                              break; }}
if (check) {
  prob=prob*2;
  printf("\nVerifier agree:prover is lying with probability
1/%d\n",prob);
  if (prob<PROBOK) goto m1; }
else printf("STOP:Prover is lying!\n");
/*~~~END OF VERIFIER ZONE~~~*/
printf("End of protocol ZKIP");
}
/*-----------------------------------------------*/
```

```
randbit(vector x)
{ /* gives 0 or 1 based on sem if sem>>10000*/
x[0]=1;
if (sem[1]>=5000) x[1]=1;
else x[1]=0;
}
/*-------------------------------------------------------*/
quadres(vector w,vector y)
{ /*computes y, the quadratic residue of w mod PQ */
apowerb(w,two,pq,y);
}
/*-------------------------------------------------------*/
select(vector w)
{ /* selects a random number aux in Z* (pq) */
vector aux;
do {
  random(pq,w);
  euclid(pq,w,aux); }
while ((aux[0]!=1)||(aux[1]!=1));
}
/*-------------------------------------------------------*/
random(vector x,vector y)
/* gives a random number y between 0 and x-1 (x<=PRI) */
{
/* Xn+1=(ASEM*Xn + CSEM) mod (2^1279 -1)=PRI */
/* ASEM^2 ~ PRI      PRI/CSEM ~ 5 */
mul(asem,sem,sem);
addsub(sem,csem,1,sem);
apowerb(sem,one,pri,sem);
apowerb(sem,one,x,y);
}
/*-------------------------------------------------------*/
int sign(int x)
{
if (x==0) return(0);
else return((abs(x)/x));
}
/*-------------------------------------------------------*/
fput_in(strng txt,vector a)
{
char aux[500];
int flag=1,len,cp,j,k;
int a0,m,i=0;
FILE *f;
while ((f=fopen(txt,"r"))==NULL) {
  printf("incorrect file\n");
  printf("filename:%s",txt); }
if (fgets(aux,500,f)==NULL) *aux='\0';
fclose(f);
while (aux[i]==' ') i++;
if (aux[i]=='-') {
  flag=-1;
  i++;
  while (aux[i]==' ') i++; }
while (aux[i]=='0') i++;
len=strlen(&(aux[i]));
```

```
if ((aux[i]=='\n')||(aux[i]=='\0')) {
  if (i==0) a[0]=0;
  else {
    a[0]=1;
    a[1]=0; }} /* supposed to be zero */
else {
  k=(len-1)/4 +1;
  a[0]=flag*k; /* 4 is log10(base) */
  if (len % 4) j=i+ len % 4;
  else j=i+4;
  do {
    cp=aux[j];
    aux[j]='\0';
    a[k--]=atoi(&(aux[i]));
    i=j;
    aux[i]=cp;
    j+=4; }
  while (k>0); }
strcpy(txt,"\n>");
put_out(txt,a);
}
/*-------------------------------------------------------------*/
fput_out(strng txt,vector a)
{
/*double l=0.434294481;*/
int a0,i,j,s,sa,t,u,v;
FILE *f;
f=fopen(txt,"w");
a0=a[0];
sa=sign(a0);
a0=abs(a0);
if (a0==0) fprintf(f,"%s","0");
else {
  if (sa<0) fprintf(f,"%s","-");
  fprintf(f,"%d",a[a0]);
  for (i=(a0-1);i>=1;i--) {
    if (a[i]<1000) fprintf(f,"%s","0");
    if (a[i]<100) fprintf(f,"%s","0");
    if (a[i]<10) fprintf(f,"%s","0");
    fprintf(f,"%d",a[i]); }}
fclose(f);                /* v=(int) floor(l*log((double)
base)+0.5);*/
}
/*-------------------------------------------------------------------
----*/
put_in(strng txt,vector a)
{
char aux[500];
int flag=1,len,cp,j,k;
int a0,m,i=0;
printf("%s",txt);
gets(aux);
while (aux[i]==' ') i++;
if (aux[i]=='-') {
  flag=-1;
  i++;
  while (aux[i]==' ') i++; }
while (aux[i]=='0') i++;
len=strlen(&(aux[i]));
```

```
if ((aux[i]=='\n')||(aux[i]=='\0')) {
  if (i==0) a[0]=0;
  else {
    a[0]=1;
    a[1]=0; }) /* suppose to be zero */
else {
  k=(len-1)/4 +1;
  a[0]=flag*k; /* 4 is log10(base) */
  if (len % 4) j=i+ len % 4;
  else j=i+4;
  do {
    cp=aux[j];
    aux[j]='\0';
    a[k--]=atoi(&(aux[i]));
    i=j;
    aux[i]=cp;
    j+=4; }
  while (k>0); }
strcpy(txt,"\n>");
put_out(txt,a);
}
/*-----------------------------------------------------*/
put_out(strng txt,vector a)
{
/*double l=0.434294481;*/
int a0,i,j,s,sa,t,u,v;
printf("%s",txt);
a0=a[0];
sa=sign(a0);
a0=abs(a0);
if (a0==0) printf("%s","0");
else {
  if (sa<0) printf("%s","-");
  printf("%d",a[a0]);
  for (i=(a0-1);i>=1;i--) {
    if (a[i]<1000) printf("%s","0");
    if (a[i]<100) printf("%s","0");
    if (a[i]<10) printf("%s","0");
    printf("%d",a[i]); })
                          /* v=(int) floor(l*log((double)
base)+0.5);*/
}
/*-----------------------------------------------------*/
red(vector c)
{
int c0,s,i,j;
c0=c[0];
s=sign(c0);
c0=abs(c0);
c[c0+1]=0;
while ((c0>0) && (c[c0]==0)) {
       c0=c0-1;
       if (c[c0]<0) for (i=0;i<=c0;i++) c[i]=-c[i]; }
```

```
for (j=1;j<=2;j++) {
  for (i=1;i<=c0;i++) {
    if (c[i]>=base) {
      c[i+1]=c[i+1]+1;
      c[i]=c[i]-base; }
    else if (c[i]<0) {
          c[i+1]=c[i+1]-1;
          c[i]=c[i]+base; }))
c0=c0+1;
while ((c0>0) && (c[c0]==0)) c0=c0-1;
c[0]=sign(c[0])*c0;
}
/*-----------------------------------------------------------*/
let(vector a,vector b)
{
int s,i;
s=abs(b[0]);
for (i=0;i<=s;i++) a[i]=b[i];
}
/*-----------------------------------------------------------*/
addsub(vector a,vector b,int sgn,vector c)
{
int sa,sb,q,r,max,min,i,s;
sa=a[0];
sb=b[0]*sgn;
q=abs(sa);
max=q;
r=abs(sb);
min=r;
if (min>max) {
  max=r;
  min=q; }
s=sign(sa)*sign(sb);
for (i=1;i<=min;i++) c[i]=a[i]+s*b[i];
for (i=min+1;i<=max;i++) {
  if (max==q) c[i]=a[i];
  else c[i]=s*b[i]; }
c[0]=sa;
c[max+1]=0;
if (sa==0) let(c,b);
if (s!=0) c[0]=max*sign(sa);
red(c);
}
/*-----------------------------------------------------------*/
mul(vector a,vector b,vector c)
{
typedef int longvector[lvsize];
vector aa,bb;
longvector cc;
int m,n,p,sa,sb,sc,i,k,s,m2,n2,p2,m21,n21,mn;
m=a[0];
sa=sign(m);
m=abs(m);
n=b[0];
sb=sign(n);
n=abs(n);
p=m+n;
sc=sa*sb;
m2=m*2;
n2=n*2;
```

```
p2=p*2;
m21=m2-1;
n21=n2-1;
mn=m2+n21;
for (i=1;i<=m;i++) {
  s=a[i];
  aa[2*i]=s / sqrtb;
  aa[2*i-1]=s % sqrtb; }
for (i=1;i<=n;i++) {
  s=b[i];
  bb[2*i]=s / sqrtb;
  bb[2*i-1]=s % sqrtb; }
for (i=0;i<=lvsize;i++) cc[i]=0;
if (m<=n) {
  for (i=1;i<=m21;i++) {
    s=0;
    for (k=1;k<=i;k++) {
      s=s+aa[k]*bb[i+1-k];
      if (s>=base) {
        s=s-base;
        cc[i+2]=cc[i+2]+1; }}
    cc[i]=cc[i]+s; }
  for (i=m2;i<=n2;i++) {
    s=0;
    for (k=1;k<=m2;k++) {
      s=s+aa[k]*bb[i+1-k];
      if (s>=base) {
        s=s-base;
        cc[i+2]=cc[i+2]+1; }}
    cc[i]=cc[i]+s; }
  for (i=n2+1;i<=mn;i++) {
    s=0;
    for (k=i-n21;k<=m2;k++) {
      s=s+aa[k]*bb[i+1-k];
      if (s>=base) {
        s=s-base;
        cc[i+2]=cc[i+2]+1; }}
    cc[i]=cc[i]+s; }}
else {
  for (i=1;i<=n21;i++) {
    s=0;
    for (k=1;k<=i;k++) {
      s=s+bb[k]*aa[i+1-k];
      if (s>=base) {
        s=s-base;
        cc[i+2]=cc[i+2]+1; }}
    cc[i]=cc[i]+s; }
  for (i=n2;i<=m2;i++) {
    s=0;
    for (k=1;k<=n2;k++) {
      s=s+bb[k]*aa[i+1-k];
      if (s>=base) {
        s=s-base;
        cc[i+2]=cc[i+2]+1; }}
    cc[i]=cc[i]+s; }
```

```
  for (i=m2+1;i<=mn;i++) {
    s=0;
    for (k=i-m21;k<=n2;k++) {
      s=s+bb[k]*aa[i+1-k];
      if (s>=base) {
        s=s-base;
        cc[i+2]=cc[i+2]+1; ))
    cc[i]=cc[i]+s; ))
for (i=1;i<=p2;i++) {
  cc[i+1]=cc[i+1]+cc[i] / sqrtb;
  cc[i]= cc[i] % sqrtb; }
for (i=1;i<=p;i++) c[i]=cc[2*i-1]+sqrtb*cc[2*i];
if (c[p]==0) p=p-1;
c[0]=p*sc;
}
/*-------------------------------------------------------*/
quot(vector a,vector b,vector q,vector r)
{
vector one,qk,aa,bb,cc,qq;
int m,n,s,t,i,j,k,l,p,t1,j1,sa,sb,u;
double ar,br,qr;
for (i=0;i<=vsize;i++) qq[i]=0;
m=a[0];
n=b[0];
sa=sign(m);
sb=sign(n);
s=abs(m);
t=abs(n);
l=s-t+1;
if (l<=0) l=1;
if (s<t) u=t;
else u=s;
if (n==0) {
  printf("\nDivision by zero attempted in QUOT\n");
  l=0;
  goto q3; }
one[0]=1;
one[1]=1;
qk[0]=1;
t1=t+1;
b[0]=t;
let(aa,a);
aa[0]=s;
if (t==1) br=(double) b[1];
else br=(double) b[t] + ((double) b[t-1])/((double) base);
q1:
if (s<t) goto q3;
if (s==t) {
  addsub(aa,b,-1,cc);
  if (cc[0]<0) goto q3; }
j=aa[0]-t;
if (s<=1) ar=(double) aa[s];
else ar=(double) aa[s] + ((double) aa[s-1])/((double) base);
qr=ar/br;
if ((qr<1.) && (j==0)) goto q3;
else if (qr<1.) {
       qr=qr*((double) base);
       j=j-1;
       if (qr<1.) qr=1.; }
k=(int) floor(qr);
```

```
qk[1]=k;
qq[j+1]=qq[j+1]+k;
q2:
mul(qk,b,bb);
for (i=t1;i>=1;i--) bb[i+j]=bb[i];
for (i=1;i<=j;i++) bb[i]=0;
bb[0]=bb[0]+j;
addsub(aa,bb,-1,cc);
s=cc[0];
p=s;
if (p<0) {
  qk[1]=qk[1]-1;
  qq[j+1]=qq[j+1]-1;
  if (qk[1]==0) {
    qk[1]=b1;
    qq[j]=qq[j]+qk[1];
    j=j-1; }
  goto q2; }
for (i=0;i<=p;i++) aa[i]=cc[i];
goto q1;
q3:
b[0]=n;
if (qq[l]==0) l=l-1;
qq[0]=l*sa*sb;
let(q,qq);
if (l==0) aa[0]=m;
else aa[0]=aa[0]*sa*sb;
if ((sa*sb<0) && (aa[0]!=0)) {
  addsub(q,one,-1,q);
  addsub(aa,b,sb,aa); }
let(r,aa);
}
/*-------------------------------------------------------------*/
euclid(vector a,vector b,vector d)
{
vector aa,bb,q,r;
int a0,b0,i,r0,aa0,bb0;
a0=abs(a[0]);
b0=abs(b[0]);
let(aa,a);
aa[0]=a0;
let(bb,b);
bb[0]=b0;
while (bb[0]>0) {
  quot(aa,bb,q,r);
  let(aa,bb);
  let(bb,r); }
let(d,aa);
}
/*-------------------------------------------------------------*/
apowerb(vector a,vector b,vector n,vector r)
{
vector two,par,aa,bb,p,s,gb;
p[0]=1;
p[1]=1;
two[0]=1;
two[1]=2;
let(bb,b);
```

```
if (bb[0]<=0) {
  printf("\nB<=0 attempted in APOWERB\n");
  goto a1; }
quot(a,n,gb,aa);
while (bb[0]>0) {
  quot(bb,two,gb,par);
  let(bb,gb);
  if (par[0]==1) {
    mul(aa,p,s);
    quot(s,n,gb,p); }
  mul(aa,aa,s);
  quot(s,n,gb,aa); }
let(r,p);
a1:
}
/*-------------------------------------------------------------*/
```

Fichier de résultats du programme décrit au point VII.2.

```
$ define lnk$library sys$library:vaxcrtl.olb

$ run essai

Base=10^4


m = 2^1279-1 is the basic module of the linear congruential method for
the pseudo-random generator:

>104079321946643990819252403273640855386152622472667048053191123504036
080596733602980122394417323241848124216139542310077913835662483234649O
813990660567732076292412950938922034577318334966158355047295942054768 9
811211693677147514178866962501384438260291732348885311160328538416585O
282556046662248318909188018470682222031405210266984354887329580288780 5
086973618690071172071055570316872908 7

----------
P*Q=N is the product of two big prime numbers (2^521-1 and 2^507-1).
To find P and Q knowing only N is supposed intractable:

>364615485029501136970713101143871109540079913994317049087258562868354
903436255206595580958951401147024129894416770392933752888490885711614 1
935206466329731015333310868255891307421498160980770516577211208619769 4
982564410671414409655048743803630814603635999408767997941543021251242 4
249883509672084810819482140514371199826439455235784330363168917 7

----------
Multiplicator a of the linear congruential method
a must be approximatively the radical of m (Coveyou-Greenberger).
In order to be a primitive number of m (a prime with m & the smallest
number h, with a^h = 1 (mod m) is h = m-1 (Eulerian Function)):

>555555555555555555555555555566666666666666666666220875921706093794716
375762404103004004990504111786204300430166155026589905240121409306972 8
244986300580825654459577486475372624861261468501270871

----------
Semence is X0, order of radical of m:

>555555555555555514114444444414444144144144144444444444444444144444516
449975617381717931183834400602374865941158555844702566131871308129524 4
033682389259290706560275662871806343945494986075 1

----------
c is the additive term of the congruential method.
A good choice for c is c ~ 0.2*m (not here):

>872213277902071346041462570639011665562079727297004617670555507851302
566746088721832395072195127174341016725178680177638925792182027 1

Password is:
355883901191721427507726221308632649988876936000897897184224089590363 6
019230789027338417354825036076205119794076697354056124619476211321470 0
653829493135166043803233890766156144724691108112284983122258167710153 0
079102942724779927474368581221322749758492293134934289087593542886761 0
326199151260566871298734881743434267429739950290551087202657 11

----------
```

Prover has given y:
3107777183149676567:1647416459909184572781490321382450494468277264297106916121390316213219735760401174922137724002493958195822014051231374425565954510027905070321364680395136203006403788172677865699869354527504752517727344739755286357337484232631746310091822020257273426402535340207484506025866715257998279945128299356742489404366593650171086144

****

ROUND 0

Prover gives z:
1475272997212262520034280280326459224410817479370302156286512244106712930419240583073959152952337441148741773887991126850645702004390576750504060709848000811021973147197696635684786789842676453768429210658272936147835960284585916775885355952652272952617556976114300374628289286003373275040098074278549180677038369155668489592337816279908524 8

Verifier chooses:
0

Prover gives u:
2104339522796421139166448277901169175107857707200650637369970796946069619474784709477761460122125261280303443413589580130109998674904059118097458391221137681131335419271549622542085710670519735841501511992128506421065452979415887200357695273939960905949139566597272455937096353165668999013407467302226215160240901857450602988719628709605496 5

Verifier agree:prover is lying with probability 1/2

ROUND 1

Prover gives z:
1031041308933798822446878775620846946150201624711835483554047704112773238928204283426785838840177299984719663162695222646342258283342570892335437279379157076285575279765497626654748838785303788072209373738452116842103303913048089235908835637734141828035666042755028358936418638297705755313003405188956517970256964434640960046514933745749724

Verifier chooses:
0

Prover gives u:
1971125878962891603658484889851258354399342487623418188624966571963204266726187476463288248733773794011558194639791641004122487611575078992552197691948624596467540689665935633121793738585622798671732090697734684469916442056543408229456697088377360753429535453029361953246301504100095316733978205541713847582915602866999143824030245772880006 35

Verifier agree:prover is lying with probability 1/4

ROUND 2

Prover gives z:
271933344695611725889772719167701930247309673605453919377969925764549 8
5452991076900759316024920765408923153590131622761420990738567782737173
4317236706234610482818097063528697415682657710109959214076545612252299
8143672632470999759932020182990865015972616453668303454050518781723144
9272682097933183568079066322293932941382568832939008894287938 9

Verifier chooses:
1

Prover gives v:
857315672055484369943489783060842307430536191926915072756962416744785 8
287231231500002627039874010134868706566203203581844224716260065046645 5
101614272535791684406335478925883543831309581203575683516667357397576 8
372805328697435648524667983394698500845086172228622351866843723468399
378932703344104473293155279517550739755567372201105636805695 7

Verifier agree:prover is lying with probability 1/8

ROUND 3

Prover gives z:
149889887888631583924702384539540031483404431416297436557197942716872 8
203038144225108011573628317311927042887052480461556491422639793155975 2
709277992803056691941031534860873028985209594166375779346181017349129 6
672347958441250075886534910965728284981434253876898707189162730833711 9
584507684733816619976237430331536864740663591298414210457641 64

Verifier chooses:
0

Prover gives u:
360677402649429761830773097832991919527440643732733812577908766651221 4
810954674141294111981774208135238694317047417254173332285427043189173 5
419655758096483868595749601539133206320399139641837854130186492554484
822023062696783771682293580986286061118296379184886938635616938648632 9
169841643542396833188242714644879249885255644139678332608732 72

Verifier agree:prover is lying with probability 1/16

ROUND 4

Prover gives z:
440543003603238611053587782983333187643698834129520786745563451013294 0
074387414897073691970245561291276363504771582709144896162173367830653 5
338928981593356811068080973610552852507399379932609986295131650357619 4
672613935077700965029048832157509049792220943527164591266518300264029 7
807299383613148724827189715995959717784608763912078264340807 6

Verifier chooses:
0

Prover gives u:
1145536842993229403266910036341740937345133131099859881139336847178280
8617644108968069458618864653290029804428971228734363493036514853826151771514114227283956468380392258514003102371947077240103202018046586122315092235841609450855044555566000376426332571970609240901768198650814899368581388038848787951878782315136543518639671219108271016864870421

Verifier agree:prover is lying with probability 1/32

ROUND 5

Prover gives z:
1711502249191607506320775720245342351956746499703774415453275333075610858802716941576247697505681120203089068980503554122282890095856133037845671997775847319809428604293856630177430296308217733282865041863550207448805935402940961208749007469039682721838895411975114269896237779428792991997862546394274872118533721875868918379119179751348940818

Verifier chooses:
0

Prover gives u:
2150528975309894497489431189106051206971296774781017000031891656601413138862963141125408995693761305639138015552962904105746009306776894255645783174140801416858585973304267563481307512241993368466537038173056010137098604864345111100729363565521590619609904788479396583693957503555062864577083145016123491372545294044679400820255473226178803381

Verifier agree:prover is lying with probability 1/64

ROUND 6

Prover gives z:
3269544032716792497121643960690154153560876519644583717911194146206177590565642092376111512493415991551094209002673122165052882698740121774120009168869528514049848546637813699959060324246640602494599364910739951723401400948766471489944584680298229140265330035098342117196481172316999971092423329848240318541270607704722838018579890097926415 33

Verifier chooses:
1

Prover gives v:
9427212249800396322915606693755986232943055331301240614973865490940582429163769192867358688609271918635873924897510309929861784606687859073874131727006433768602088943910673852472491263737834789773446908949771903219684389960759795729071012863287973693587232531244986418096540972306468888541322722312647119366424417717993347844558955014135441

Verifier agree:prover is lying with probability 1/128

ROUND 7

Prover gives z:
18691464930063194185516514023450702343730912000065790837140209387618
59030026345353937686502363441531168793345908497331922216046973227912
92217262736857834732517175137366550283156247468263865508448792278117
13498859064582311709912857534012133867727976542843677109820398440049
11703440491068367827674753059332137412792517513888123527863115

Verifier chooses:
1

Prover gives v:
19377001061294220777611113553945133813592976424932901843761870600276
31654053721956122392276825686641401913499590828535001319630069544279
81026752032990420629117190144721618953751280240063381052825827544044
02983981575436716112009872236370774517980395078661327776321675824599
95587307358724073640814329804134848049587543020912969219811212

Verifier agree:prover is lying with probability 1/256

ROUND 8

Prover gives z:
24970109956986656411501911440510270140931579304634486118021515103160
65364411855470839272875361403756228535401964629803678797844555935829
46963885324973486402282973833604045651588966559920662331193962740999
23927104464718342719753075694815563919485136752608013731909511167524
95600611468839460435464599281861723220949764237624637802585905

Verifier chooses:
1

Prover gives v:
32562711526680246057302390981453999735063617058464771712935936159362
73564434860903619866284629888723053920303651733299325761211571644046955
29653051190607027949618747116748251085684670810833946913382972930664
51710468702005026552253951968304474449859898602408010958430584535832
20224931958149894643855724935534072591383265533771964257469929

Verifier agree:prover is lying with probability 1/512

ROUND 9

Prover gives z:
22400530677687830197040489560752570577725688341015241982040488357715
54785022043339554916624108050661291945671019325757060359078650457577
27442207040901358413979541094258637698736842984331993333524309569798
20721028112128464596499031042835824118328502160042746600264105669062
11158336497482749762001990029630006859019513769794381178825008

Verifier chooses:
0

```
Prover gives u:
1344962377574671727641427484994698977983572238481413355748913717617905
0526891769666285292961379449730019105026962777875050875171106131575093
8197862215305042903719196207033646913693631316907134951603085092723134
3247198342529488503010600039542933872810957177946799500190357602701777
0346746608041108054516485695977409202173196197821492976872082 3

Verifier agree:prover is lying with probability 1/1024

End of protocol ZKIP

  YSTAELENS     Job terminated at 18-MAY-1988 17:15:12.10

  Accounting information:
  Buffered I/O count:              88       Peak working set size:    667
  Direct I/O count:                99       Peak page file size:     1673
  Page faults:                    970       Mounted volumes:            0
  Charged CPU time:      0 00:02:09.16       Elapsed time:      0 00:07:03.9
```

BIBLIOGRAPHIE

* * * * * * * * * * * * *

[Adleman & Huang, 87]

L.M. Adleman and M.D. Huang, "Recognizing Primes in Random Polynomial Time", Proc. of the 19th Annual ACM Symposium on the Theory of Computing, 1987, pp.462-469.

[Adleman, Pomerance & Rumely, 83]

L.M. Adleman, C. Pomerance and R.S. Rumely: "On Distinguishing Prime Number from Composite Numbers", Ann. of Math. 117, 1983, pp.173-206.

[Atlan, 79]

H. Atlan, "Entre le Cristal et la Fumée...", Seuil, 1979.

[Babai, 85]

L. Babai : "Trading group theory for randomness", Proc of the 17th Annual ACM Symposium on the Theory of Computing, 1985 pp.421-429.

[Bengio, Brassard & al., 87]

S. Bengio, G. Brassard, Y. Desmedt, C. Goutier and J.J. Quisquater : "Aspects and Importance of Secure Implementations of Identifications Systems", June 1987 (Draft).

[Bennett & Brassard, 84]

C.H. Bennett and G. Brassard : "Quantum Cryptography : public-key distribution, and coin-tossing", IEEE International Conference on Computers, Systems and Signal Processing, Bangalore, India, Dec 1984.

[Blum, 81]

M. Blum : "Three Applications of the Oblivious Transfer", Dept. of EECS, Univ. of California, Berkeley, 1981.

[Blum, 86]

M. Blum : "How To Prove A Theorem So No One Can Claim It" , 11 Aug. 86, p 3. (ISO/TC97/SC 20/WG 2 N 73).

[Brassard, Chaum & Crépeau, 87]

G. Brassard, D. Chaum and C. Crépeau: "Minimum Disclosure Proofs of Knowledge", technical report 87, CWI, Amsterdam.

[Brassard & Crépeau, 86]

G. Brassard and C. Crépeau: "Non-transitive Transfer of confidence: a perfect zeroknowledge Protocol for SAT and beyond", Proc. of FOCS 86, pp.188-195.

[Brickell, Chaum & al., CRYPTO'87]

E. Brickell, D. Chaum, I. Damgård and J. van de Graaf: "Gradual and Verifiable Release of a Secret", Proc. of Crypto 87, Springer.

[Chaum, Damgård & Crépeau, CRYPTO'87]

D. Chaum, I. Damgård and C. Crépeau: "Multiparty Unconditionally Secure Protocols", presented at rump session of Crypto 87, submitted to STOC 88.

[Chaum, Damgård & van de Graaf, CRYPTO'87]

D. Chaum, I. Damgård and J. van de Graaf: "Multiparty Computations Ensuring Privacy of each Party's Input and Correctness of the Result", Proc. of Crypto 87, Springer.

[Chaum & Evertse, CRYPTO'86]

D. Chaum and J.H. Evertse: "A Secure and Privacy Protecting Protocol for Transmitting Personal Information Between Organisations", Proc. of Crypto 86, Springer.

[Chaum, Evertse & al., CRYPTO'86]

D. Chaum, J.H. Evertse, J. van de Graaf and R. Peralta: "How to Demonstrate Possession of a Discrete Log Without Revealing it", Proc. of Crypto 86, Springer.

[Cohen & Lenstra, 87]

H. Cohen and H.W. Lenstra : "Implementation of a New Primality Test". Mat. Comp. 46, 1987.

[Cook, 71]

S. Cook : "The Complexity of theorem proving procedures", Proc. of the 3rd Annual ACM Symposiun on the Theory of Computing, 1971, pp.151-158.

[Damgård, 87]

I. Damgård : "The Application of Claw-Free Functions in Cryptography : Unconditional Protection in Cryptographic Protocols", thèse Mathematical Institute Aarhus University Denmark 1987.

[De Pra, 87]

M. De Pra : "La carte à microprocesseur CP8 - Description et évaluation", mémoire FNDP Namur Belgium 86-87.

[Desmedt, Goutier & Bengio, CRYPTO'87]

Y. Desmedt, C. Goutier and S. Bengio: "Special Uses and Abuses of the Fiat-Shamir Passport Protocol", submitted to Crypto 87, Santa Barbara, California, USA, August 1987.

[Desmedt & Quisquater, CRYPTO'86]

Y. Desmedt and J.J. Quisquater: "Public-key systems based on the difficulty of tampering (Is there a difference between DES and RSA?)", Presented at Crypto 86, Santa Barbara, California, USA, August 1986.

[Diffie & Hellman, 76]

W. Diffie and M.E. Hellman: "New Directions in Cryptography", IEEE Trans. Inform. Theory, IT-22(6), pp.644-654, November 1976.

[Davies & Price, 80]

D. Davies and W. Price: "The Application of Digital Signatures Based on Public Key Crypto-Systems", Proc. of Comp Con 1980, pp.525-530.

[Ekeland, 84]

I. Ekeland, "Le Calcul et l'Imprévu; les Figures du Temps de Kepler à Thorn", Paris Seuil, 1984.

[Feige, Fiat & Shamir, 87]

U. Feige, A. Fiat and A. Shamir: "Zero Knowledge Proofs of Identity", May 25-27, 1987. To appear in the proccedings of ACM Symp. Theory of Computing, New York, USA.

[Fiat & Shamir, CRYPTO'86]

A. Fiat and A. Shamir: "How to Prove Yourself: Practical Solutions to Identification and Signature Problems", August 11-15, 1986. Presented at Crypto 86, Santa Barbara, California.

[Fiat & Shamir, 87]

A. Fiat and A. Shamir: "Unforgeable Proofs of Identity". In Proc. Securicom 87, March 4-6, 1987. Paris, France.

[Garey & Johnson, 79]

M. Garey and D. Johnson: "Computers and Intractability. A Guide to the Theory of NP-Completeness", Freeman & Company S.F., 1979.

[Goldreich, Micali & Widgerson, CRYPTO'86]

O. Goldreich, S. Micali and A. Widgerson: "Proofs that Yield Nothing but the Validity of the Assertion", Proc. of Crypto 86, Springer.

[Goldreich, Micali & Widgerson, 86]

O. Goldreich, S. Micali and A. Widgerson : "How to play any mental game", extended abstract, 4.1 , 1986.

[Goldwasser & Killian, 86]

S. Goldwasser and J. Killian: "Almost All Primes Can Be Quickly Certified", Proc. 18th STOC, Berkeley 1986, pp.316-329.

[Goldwasser & Micali, 84]

S. Goldwasser and S. Micali : "Probabilistic Encryption", JCSS, vol 28, No.2, 1984, pp.59-68.

[Goldwasser, Micali & Rackoff, 85]

S. Goldwasser, S. Micali and C. Rackoff : "The Knowledge Complexity of Interactive Proof Systems", Proc. of STOC 85, pp.291-304.

[Goldwasser, Micali & Rackoff, 86]

S. Goldwasser, S. Micali and C.Rackoff : "The Knowledge Complexity of Interactive Proof Systems", Extended abstract 86.

[Goldwasser, Micali & Rivest, 84]

S. Goldwasser, S. Micali and R. Rivest : "A paradoxical Solution to the Signature Problem", Proc. of FOCS 1984, pp.441-448.

[Hardy & Wright, 79]

G.H. Hardy and E.M. Wright : "An Introduction to the Theory of Numbers", Fifth edition, Oxford, 1979, pp.73-78.

[Knuth, 69]

D.E. Knuth :"The Art of Computer Programming. Vol 2 : Seminumerical Algorithms", Addison-Wesley, Reading Mas. 1969, Chap 3.

[Mehlhorn, 84]

K. Mehlhorn: "Data Structures and Algorithms 2. Graph Algorithms and NP-Completeness", Springer-Verlag, 1984.

[Prigogine, 82]

I. Prigogine: "Physique, Temps et Devenir", Masson, 1982.

[Quisquater, CRYPTO'86]

J.-J. Quisquater : "Some DES Cycling Results", Crypto 86.

[Quisquater, 87]

J.-J. Quisquater : "Secret distribution of keys for public-key systems", Advances in Cryptology, Proc. of Crypto 87. Lecture Notes in Computer Science, N° 293, Springer-Verlag, 1988.

[Rabin, 76]

M.O. Rabin: "Probabilistic Algorithms", in Algorithm and Their Complexity: Recent Results and New Directions, J.F. Traub (editor), Academic Press, New York, NY, 1976, pp.21-39.

[Rabin, 81]

M.O. Rabin: "How to Exchange Secrets by Oblivious Transfer", technical Memo. TR-81, Aiken Computation Laboratory, Harvard University, 1981.

[Riesel, 85]

H. Riesel : "Prime Numbers and Computer Methods for Factorization", Appendix 7, Birkhäuser Boston-Basel-Stuttgart, 1985.

[Rivest, Shamir & Adleman, 78]

R.L. Rivest, A. Shamir & L. Adleman : "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", Comm ACM 21, 1978, pp.120-126.

[Shamir, 79]

A. Shamir: "How to Share a Secret", CACM, vol.22, No.11, 1979, pp.612-613.

[Shamir, 86]

A. Shamir : "Interactive Identification", March 23-29, 1986. Presented at the Workshop on Algorithms, Randomness and Complexity, Centre International de Rencontres Mathématiques (CIRM), Luminy (Marseille), France.

[Simmons, CRYPTO'85]

G.J. Simmons : "The secure subliminal channel (?)". In H.C. Williams, edito, Advances in Cryptology. Proc. of Crypto 85. (Lecture Note in Computer Science 218) pp.33-41, Springer-Verlag, 1986. Santa Barbara, California, August 18-22, 1985.